HNDIT2012 Fundamentals of Computer Programming

Introduction to computer programs

Computer

Programmable electronic machine which working under a given set of instruction

Computer program is a set of instructions to execute by computer

Computer Programming Language

Programming Language is a language designed to communicate with computer

- Because computer is an electronic machine, it can only understand voltage levels
  - 0 to represent lower voltage level
  - 1 to represent higher voltage level
- This language is known as machine language

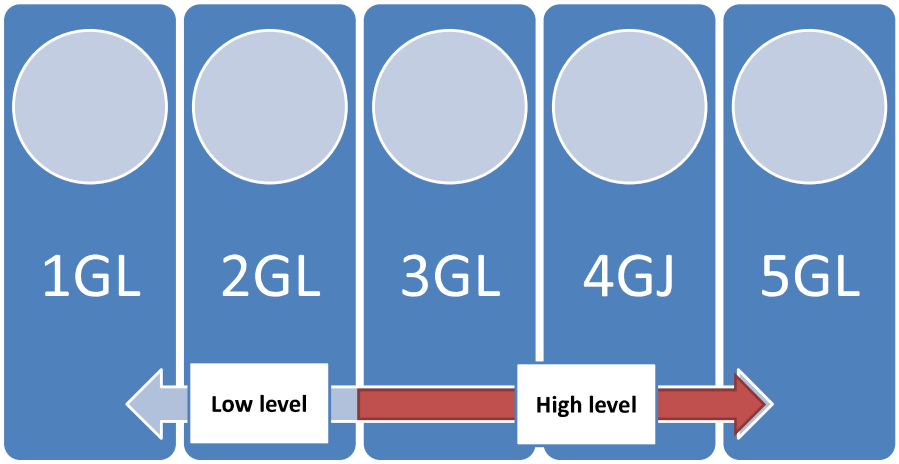
Components of a Language

- Semantics
  - Meaning
- Syntax
  - How the meaning is presented

Generation of Programming Languages

- First-Generation Languages :
  - These are low-level languages like machine language.
- Second-Generation Languages :
  - These are low-level assembly languages used in kernels and hardware drives.
- Third-Generation Languages :
  - These are high-level languages like C, C++, Java, Visual Basic, and JavaScript.
- Fourth Generation Languages :
  - These are languages that consist of statements that are similar to statements in the human language. These are used mainly in database programming and scripting, like Perl, Python, Ruby, SQL, and MatLab
- Fifth Generation Languages :
  - These are the programming languages that have visual tools to develop a program like, Mercury, OPS5, and Prolog.

Generation of Programming Languages



Translation

- **Interpreting**

  Instructions are translated and executed one after one.

  HTML,
- **Compiling**
  - Translate all the instructions of the program in a one step and create an executable file.
  - Some languages create intermediate form known byte code (java)

Translators

Interpreter

- converts source code statements into equivalent machine language statements and execute.
- Source program is required for every execution
- Slow execution

Compiler

- Once all errors have been corrected translates the entire source code as a single unit.
- After the compilation process the source program is no more required
- Faster execution

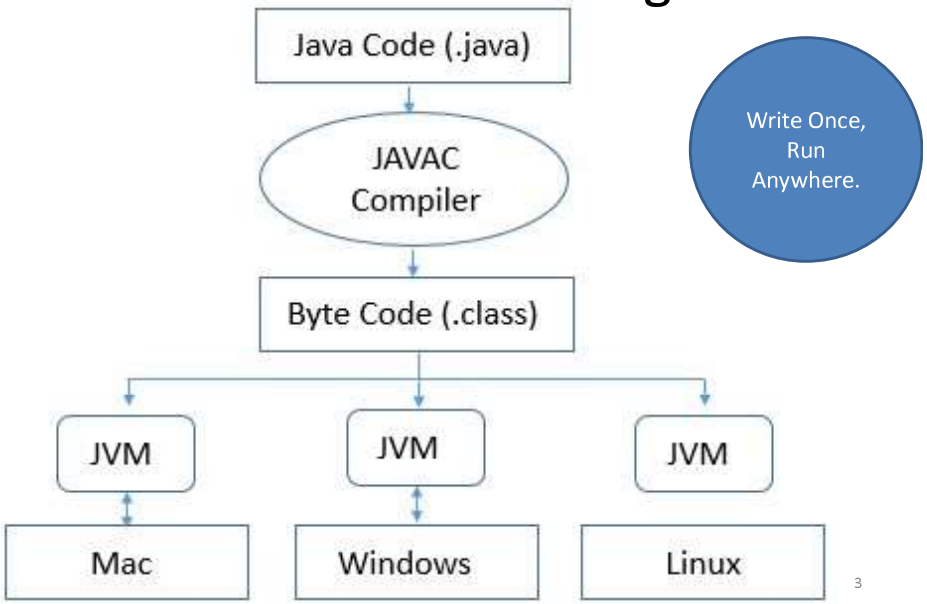
HNDIT2012 Fundamentals of Computer Programming

Introduction to computer programming in Java

How Java is working

- Source Code
  - Source code is plain text
  - File name extension is .java
  - Compiler is Javac
  - Object code/byte code extension is .class

How Java is working

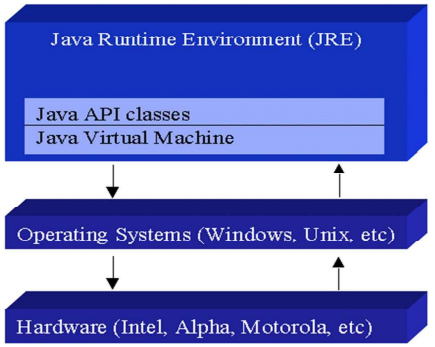


Write Once, Run Anywhere.

Java Virtual Machine

- Collection of software components that enables a computer to run a Java program
- Lods , Versifies and execut code
- Interpreter which provides runtime environnent

Virtual machine is a software that emulate a computer system, which provide functionality of a physical computer



Java Features

- **Case sensitive**
  - Compiler recognize the deferent between simple and capital letters
- **Secure**
  - When this byte codes are executed , the JVM can take care of the security.
- **Robust**
  - Strong memory management
  - Exception handling mechanism
  - Type checking mechanism
- **Portable**
  - Program written in Java can be run on deferent computing environments.
- **Multithreaded**
  - Java supports parallel processing through execution of multiple threads

HNDIT2012 Fundamentals of Computer Programming

Java Hello World

First Java Program

```
class HelloWorldApp
{
   public static void main(String[] args)
   {
      System.out.println("Hello World!"); // Display the text.
   }
}
```

- Save with **.java** extension
- If a public class is present, the class name should match the file name

Hello World

•**class** keyword is used to declare a class in java.
•**public** keyword is an access modifier which represents visibility, it means it is visible to all.
•**static** is a keyword is used to create static method. no need to create object to invoke the static method
•**void** is the return type of the method
•**main** represents startup of the program.
•**String[] args** is used for command line argument.
•**System.out.println()** is used print statement.

Comments

- Statements which are ignored by compiler
- Used to add explanation of code
- Make the source code is easier for humans to understand (increase the readability of code)

Types of Java Comments

- Line comments
  - // Comments a line of code
- Block comments
  - /* */ Comments a block of code

Escape Sequences

Notation	Character represented
\n	Newline
\r	Carriage return
\b	Backspace
\s	Space
\t	tab
\"	Double quote
\'	Single quote
\\	backslash

HNDIT2012 Fundamentals of Computer Programming

Introduction to IDE

IDE

a software application that helps programmers develop software code efficiently.

- IDE increases developer productivity by combining capabilities such as software editing, building, testing, and packaging in an easy-to-use application.

Features of an IDE

- Code editor
  - Write and edit code
- Build automation
  - Compiling computer source code into binary code
  - Packaging binary code
  - Running automated tests

Features of an IDE

- Form Designer
  - Allows a programmer to build forms with a drag and drop interface.
- Debugger
  - Test and find bugs (errors)
- Refactoring
  - Restructuring existing computer code without changing its external behavior

Features of an IDE

- Syntax highlight:
  - The code editor provides a feature to colour different text depending on the function that it carries out

Popular Java IDE

NetBeans.IDE

eclipse

ORACLE JDEVELOPER

IntelliJ IDEA

drjava

BlueJ

NetBeans

NetBeans is an integrated development environment (IDE) for Java. NetBeans allows applications to be developed from a set of software components called modules. NetBeans runs on Windows, macOS, Linux and Solaris. In addition to Java development, it has extensions for other languages like PHP, C, C++, HTML5 and JavaScript.

Installing NetBeans

- Visit https://netbeans.apache.org/download/index.html

Features of NetBeans IDE

- Easy to install with easy configurations
- Code Editing.
- Graphical User Interface.
- Debugging
- Form designer

Debugging

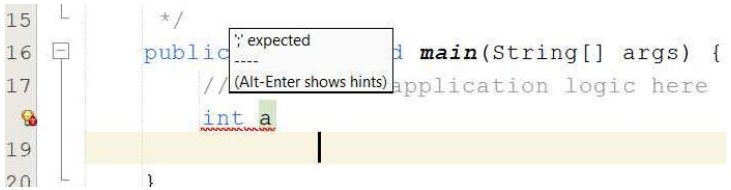
The process of identifying and removing errors from computer hardware or software.

Types of errors

- Syntax errors
  - Syntax errors are detected at compile-time
  - Intelligent IDE identify syntax errors in coding time
- Logical errors
  - Error in code logic to produce an unexpected output
- Run time error

Syntax errors in NetBeans

- NetBeans IDE find the syntax errors
- Underline the error with wavy red line.
- Display an explanation of the error when mouse pointer is moving the on the wavy red line.



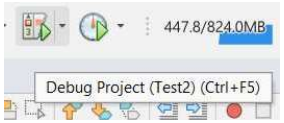
Debugging in NetBeans

- Setting Breakpoints
  - Breakpoint tells the debugger where to stop during execution

```
    public static void main(String[] args) {
Line Breakpoint// TODO code application logic here
        int a=5/2;
          System.out.println(a);
    }
}
```


Debugging in NetBeans

- Start debugging
  - Execution will halt at breakpoint



Debugging in NetBeans

- Step over
  - Run to next line
- Step into
  - Enter to a method and stop at the first line
- Step out
  - Run to method completion

Debugging in NetBeans

- Variable window
  - Display the values of variables

HNDIT2012 Fundamentals of Computer Programming

Introduction to IDE

Software Versioning

Software versioning

Software versioning is the process of assigning either unique version names or unique version numbers to unique states of computer software.

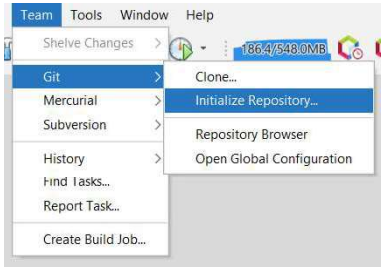
- Within a given version number category, these numbers are generally assigned in increasing order and correspond to new developments in the software.

Versioning with NetBeans

Apache NetBeans IDE provides support for the Git, Mercurial and Subversion version control systems, which are distributed version control and source code management systems

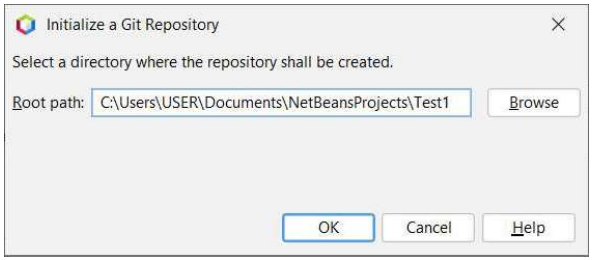
Git versioning with NetBeans

- Initialize a repository



Initialize a repository

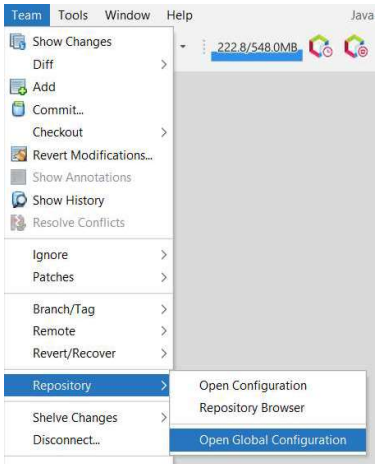
Repository allows developer to save versions of the code, which can access when needed.



A .git subfolder is created in current working directory, which all the project snapshots are stored. This will also create a new main branch.

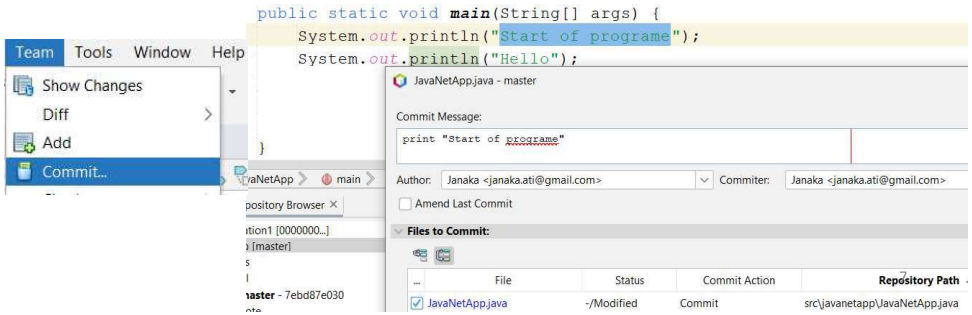
C:\Users\USER\Documents\NetBeansProjects\Test1\.git

Global configuration



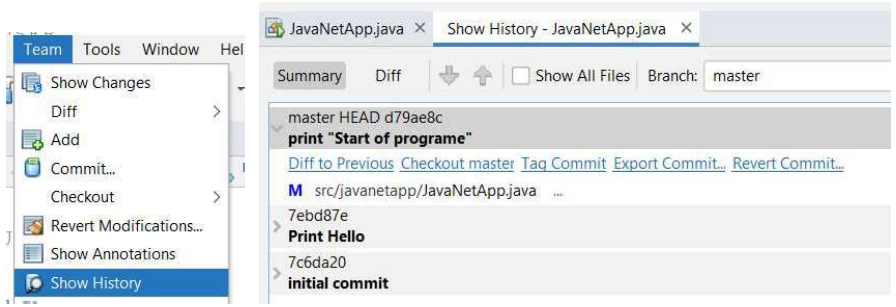
Commit

- Commit means snapshots or milestones along the timeline of a Git project.
- So, developer can do time to time changes in project and commit with a suitable message.



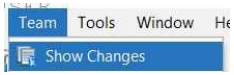
View Commit history

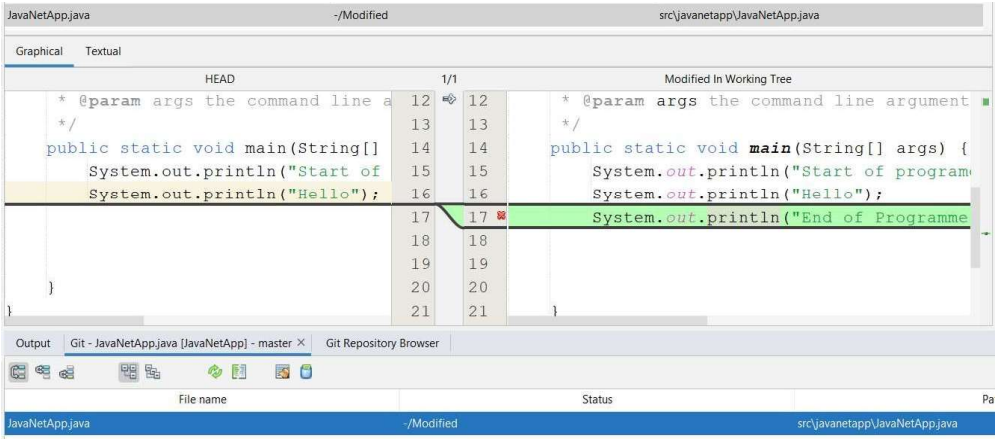
- Display the commit history



Show changes

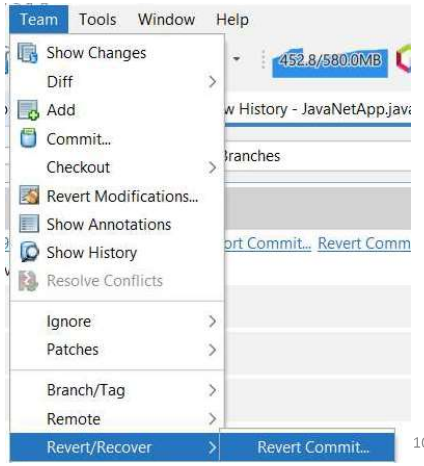
- Display the changes in project





Revert commit

- Reverse back to previous commit



HNDIT2012 Fundamentals of Computer Programming

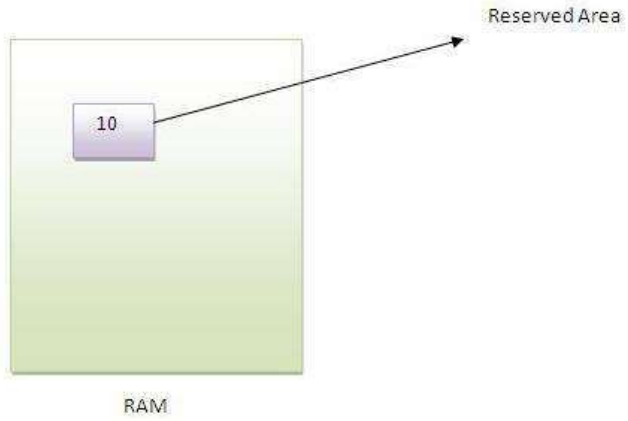
Variables and Data Types

Terminology

- Key word – special reserved word which is having a meaning to compiler. So key words cant be used for identifiers
- Identifiers - word used by a programmer to name a variable, method, class or label

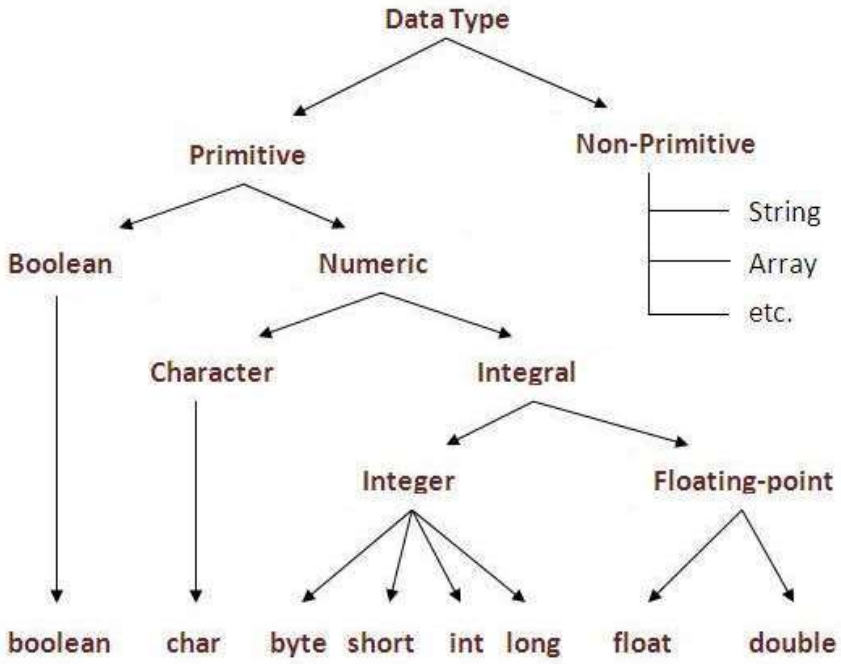
Variable

- Variable is a location in computer memory which can hold data.



Data type

- Classification of data (value of a variable) which tells the compiler or interpreter how the programmer intends to use the data.



Primitive data types

- **byte**:
  - 8-bit signed two's complement integer
  - -128 to 127 (inclusive)
  - Default value is 0.
- **short**:
  - 16-bit signed two's complement integer
  - -32,768 to 32,767 (inclusive)
  - Default value is 0.
- **int**:
  - 32-bit signed two's complement integer
  - -2,147,483,648 to 2,147,483,647 (inclusive)
  - Default value is 0

Primitive data types

- **long**:
  - 64-bit signed two's complement integer
  - -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (inclusive).
- **float**:
  - Single precision 32-bit IEEE 754 floating point.
- **double**:
  - Double precision 64-bit IEEE 754 floating point.
- **boolean**:
  - Two possible values: true and false
- **char**:
  - Single 16-bit Unicode character
  - '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).

Creating and Accessing Variables

- int A;
- A=10;
- char B;
- B=57;

Variable/Identifier Naming

- Java keywords cant be used as variable names
- Variable names are case sensitive
- All identifiers should begin with a letter (A to Z or a to z), currency character ($),underscore (_)
- Subsequent characters may be letters, digits, dollar signs, or underscore characters.
  - legal identifiers:
    - age, $salary, _value, __1_value
  - Iillegal identifiers:
    - 123abc, -salary

Variable declaration

- Informing the compiler about the variables that a program is going to use is variable declaration
- Declaration inform three things
  - Variable name, Data type and access level

Syntax
- *Type identifier;*
- *Access_modifier Type identifier;*

Example

```
int a;
int a, b, c; // Declares.
private int a;
```

Variable initialization

- Initialization
  - Setting an initial value to a variable

Syntax
- *Type identifier=initial value;*
- *Access_modifier Type identifier=initial value;*

Example

```
int a = 10, b = 10; // initialization
double pi = 3.14159;
Public double pi = 3.14159;
```

Literal

- The value stored in a variable

Variables in calculations

```
class TestVar
{
        public static void main(String[] args)
        {
        int a=2,b=4,c;
        c=a+b;
        System.out.println("Total is : "+ c);
        }
}
```

Types of Variables

- **Local Variable**
  - A variable that is declared inside the method is called local variable.
- **Instance Variable**
  - A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static. So values of **Instance Variable** are unique to each *instance* of a class
- **Static variable**
  - A variable that is declared as static is called static variable. It cannot be local. There is exactly one copy of this variable, regardless of how many times the class has been instantiated
- **Parameters**
  - Variables which pass values with method calls

Type conversion / Type casting

Changing a value from one data type to another type is known as data type conversion

- Implicit conversion
  - Automatic conversion by JVM
- Explicit conversion
  - Not automatically done by JVM
  - Programmer need to specify the target type in parentheses

Widening Conversion

- Value of lower size data type converted to a value of a higher size data type without loss of information
- This is done implicitly by the JVM and also known as implicit casting

```
int a = 100;
double b = a;
System.out.println(b);
//integer (4 Byte) converted into double (8 Byte)
```

Widening Conversion

- byte can be converted to short, int, long, float, or double
- short can be converted to int, long, float, or double
- char can be converted to int, long, float, or double
- int can be converted to long, float, or double
- long can be converted to float or double
- float can be converted to double

Narrowing Conversion

- value of higher size data type converted to a value of a lower size data type which can result in loss of information
- This is not done implicitly by the JVM and requires explicit casting.

```
double a = 100.7;
int b = (int) a;
System.out.println(b);
```

Narrowing Conversion

- short can be converted to byte or char
- char can be converted to byte or short
- int can be converted to byte, short, or char
- long can be converted to byte, short, or char
- float can be converted to byte, short, char, int, or long
- double can be converted to byte, short, char, int, long, or float

Character functions

SN	Methods with Description
1	isLetter() Determines whether the specified char value is a letter.
2	isDigit() Determines whether the specified char value is a digit.
3	isWhitespace() Determines whether the specified char value is white space.
4	isUpperCase() Determines whether the specified char value is uppercase.
5	isLowerCase() Determines whether the specified char value is lowercase.
6	toUpperCase() Returns the uppercase form of the specified char value.
7	toLowerCase() Returns the lowercase form of the specified char value.
8	toString() Returns a String object representing the specified character value that is, a one-character string.

Numeric functions

Sr.No	Methods with Description
1	**abs()** This method returns the absolute value of a double value.
2	**cbrt()** This method returns the cube root of a double value.
3	Sqrt() Return the square root
4	pow(x,y) Return the power
5	min(x,y) Return the minimum
6	max(x,y) Return the maximum
7	ceil(x) Returns the double value that is greater than or equal to the nearest integer
8	Floor(x) Returns the double value that is smaller than or equal to the nearest integer

HNDIT2012 Fundamentals of Computer Programming

Expressions and Operators

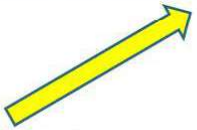
OPERATORS

Operators

- Assignment Operators
- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators

Assignment Operator (=)

```
A=15;
num1 = num2 = num3 = A + 5;
```



Right side of Assignment operator is evaluated first and then the Assignment takes place

Arithmetic operators

Operator	Description
+	Additive operator (also used for String concatenation)
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Remainder operator
++	Increment operator
--	Decrement operator

Operator Precedence

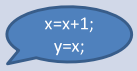
*	/	%
+	-	

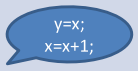
When operators of equal precedence appear in the same expression they are evaluated from left to right;

Increment Operator

Pre Increment	Post Increment

int x,y;
 	
x= 5;
 	
y = ++x;
 	
System.out.println (“x = “ + x);
 	
System.out.println (“y = “ + y);
	int x,y;
	
	x= 5;
	
	y = x++;
	
	System.out.println (“x = “ + x);
	
	System.out.println (“y = “ + y);





Write the output

```
int a, b, c;
a = b = c = 1;
a = ++b + c;
System.out.println("a = " + a + ", b = " + b + ", c = " + c);
b = a++ - 1;
System.out.println("a = " + a + ", b = " + b + ", c = " + c);
c = a + a++ + --b + b;
System.out.println("a = " + a + ", b = " + b + ", c = " + c);
```

Relational Operators

Operator	Meaning
!=	not equal to
==	is equal to
<	less than
>	greater than
>=	greater than or equal to
<=	less than or equal to

Which of the following are relational expressions in Java?

- x == 3
- x = 3
- x >= 3
- x * 3
- 3 < x
- x - 3 <= 10

For each of the following statements, assign variable names for the unknowns and rewrite the statements as relational expressions.

- A customer's age is 65 or more.
- The temperature is less than 0 degrees.
- A person's height is over 6 feet.
- The current month is 12 (December).
- A person is older than 55 or has been at the company for more than 25 years.
- A width of a wall is less than 4 metres but more than 3 metres.
- An employee's department number is less than 500 but greater than 1, and they've been at the company more than 25 years.

Logical Operators

Operator (Boolean Operators)	Operator (Short Circuit Operators Condition Operator)	Meaning
&	&&	Logical AND
|	||	Logical OR
!		Logical NOT

- short circuit logical operators evaluate second expression only if that is needed.

Given that a = 5, b = 2, c = 4, and d = 5, what is the result of each of the following Java expression?

- a == 5
- b * d == c * c
- d % b * c > 5 || c % b * d < 7
- d % b * c > 5 && c % b * d < 7

Bitwise Operators

- **Acts on individual bits of an integer**

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
>>	Right Shift
<<	Left Shift

p	q	p&q
1	1	1
1	0	0
0	1	0
0	0	0

p	q	p|q
1	1	1
1	0	1
0	1	1
0	0	0

p	q	p^q
1	1	0
1	0	1
0	1	1
0	0	0

p	~p
1	0
0	1

Shift Operators

<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.

```
class Test {
 public static void main(String args[]) {
    int a = 60;
    int b = 13;
    int c = 0;
    c = a & b;
    System.out.println("a & b = " + c );
    c = a | b;
System.out.println("a | b = " + c );
    c = a ^ b;
System.out.println("a ^ b = " + c );
    c = ~a;
System.out.println("~a = " + c );
    c = a << 2;
    System.out.println("a << 2 = " + c );
    c = a >> 2;
System.out.println("a >> 2  = " + c );
}
}
```

Operator Precedence

Operators	Precedence
Parentheses	()
unary	negative (-), logical NOT (!)
multiplicative	* / %
additive	+ -
relational	< > <= >=
equality	== !=
logical AND	&&
logical OR	||

Exercise

- 1.Write a program to convert the distance given in m to km and m
- 2.Write a program to convert temperature in Fahrenheit in to Celsius

  $C = (F - 32) \times 5/9$

  Test your code for following values
  - 45 F = 7.22 C
  - -87 F = -66.11 C
- 3.Write a program to print individual digits of a 3 digit number
- 4.Calculate the area of a circle

  $$A = \pi r^2$$

  - Math.**PI will give the value of** π
  - You have to import
    - import **java**.lang.Math.*;
- 5.Calculate the roots of a quadratic equation. Assign values for a,b,c

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

HNDIT2012 Fundamentals of Computer Programming

Control Flow Statements

Control flow

control flow (or **flow** of **control**) is the order in which individual statements of an **program** are executed .

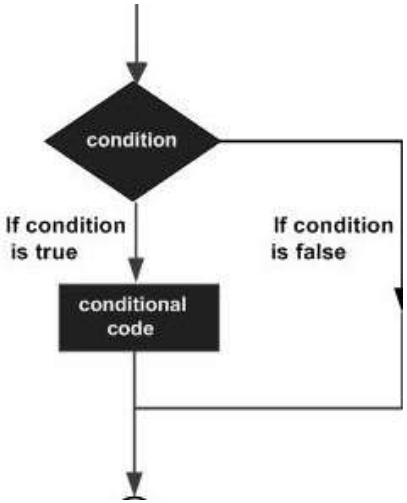
Control flow structures

- Sequential structure :
  - Default mode. Statements are executed from top to bottom of program.
- Selection structure:
  - **Selection structure** or conditional **structure**, is different blocks of code are performed based on whether a boolean condition is true or false
  - If-else, switch-case
- Repetition structure:
  - Same block of code is executed again and again based on whether a boolean condition is true or false
  - while, do-while, for

If condition

- Syntax

```
If(expression)
{
statement(s)
}
Else
{
statement(s)
}
```



If condition

Syntax

```
If(expression) {    statement(s)} else{statement(s)} }
```

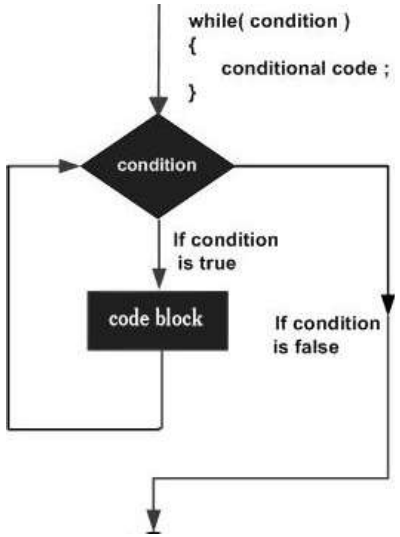
Example

```
public static void main(String[] args)
  {
    int Marks=92;
    if (Marks<40 )
        System.out.print("Result : Fail");
    else
        {
        if (Marks<60 )
                System.out.print("Result : Simple Pass");
            else
                System.out.print("Result : Credit Pass");
        }
}
```

while loop

- Syntax

```
while (expression)
{
statement(s)
}
```



while loop

Syntax

```
while (expression) {    statement(s)}
```

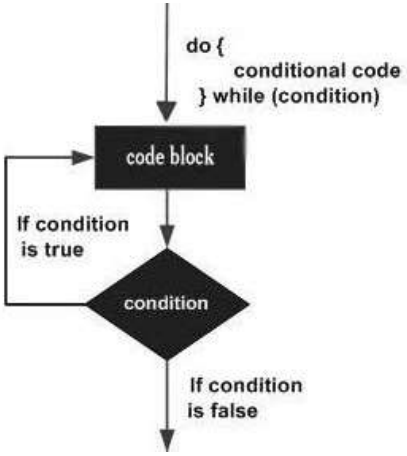
Example

```
public static void main(String[] args)
  {
        int count = 1;
        while (count < 11)
        {
                System.out.println("Count is: " + count);
                count++;
        }
}
```

Do while loop

- Syntax

```
do
 {
statement(s)
}
 while (expression)
```



do while loop

Syntax

```
do {    statement(s)} while (expression)
```

Example

```
public static void main(String[] args)
  {
        int count = 1;
        do
        {
              System.out.println("Count is: " + count);
              count++;
        }
        while (count < 11)
 }
```

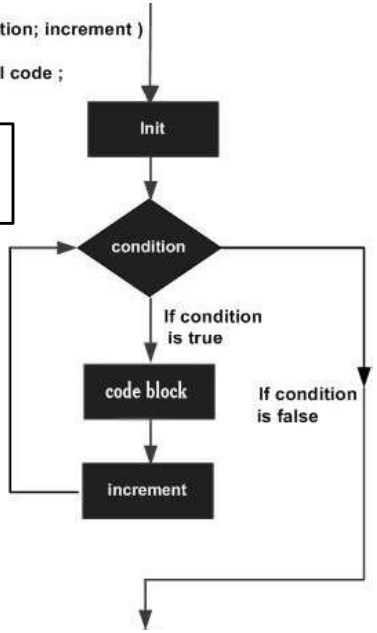
statements within the do block are always executed at least once

For loop

- Syntax

```
for(initialization; expression; change)
 {}
```

- Change can be increment or decrement



for loop

Syntax

```
for(initialization; expression; change) {}
```

Example

```
public static void main(String[] args)
  {
        for(count=0;count < 10;count++) )
              {
               System.out.println("Count is: " + count);
              }
}
```

- The ***initialization*** expression initializes the loop; it's executed once, as the loop begins.
- When the ***termination*** **expression** evaluates to false, the loop terminates.
- The ***change*** expression (increment/decrement) is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment *or* decrement a value

For-Each Loop

- Used to loop through elements in an **array**
- Syntax

```
for (type variableName : arrayName)
 {
  // code block to be executed
}
```

- Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

break statement

- Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
- The break statement has two forms:
  - labeled and unlabeled.
- Unlabeled break
  - Break out from a case
  - Terminate a for, while, or do-while loop

break statement

Example

```
class Test
 {
 public static void main(String[] args)
   {
          int count ;
          for(count=0;count < 10;count++)
                {
                if (count==5)
                 {break;}
                System.out.println("Counting: " + count);
                }
   }
 }
```

Continue statement

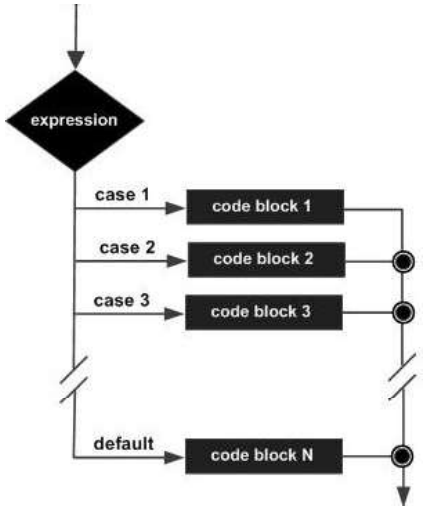
- Continue statement skips the current iteration of a for, while , or do-while loop

```
class Test
  {
  public static void main(String[] args)
      {
       int count ;
       for(count=0;count < 10;count++)
            {
            if (count==5)
            {continue;}
          System.out.println("Counting: " + count);
            }
      }
  }
```

switch Statement

- Syntax

```
switch ( expression )
{
   case value:
         statement(s);
         break;
   case value:
         statement(s);
         break;
   default:
         statement(s);
}
```



- keyword **break** is needed to break out of each case.
- **default** will execute, only if the execution skip from all the cases

switch Statement

Example

```
public static void main(String[] args) {
       int Day = 4;
      String DayString;
      switch (Day) {
         case 1:  DayString = "Sunday";     break;
         case 2:  DayString = "Monday";     break;
         case 3:  DayString = "Tuesday";    break;
         case 4:  DayString = "Wednesday";  break;
         case 5:  DayString = "Thursday";   break;
         case 6:  DayString = "Friday";     break;
         case 7:  DayString = "Saturday";   break;
         default: DayString = "Invalid Day"; break;
      }
      System.out.println(DayString);
   }
}
```

if	switch
• Test many variables • Test any data type • Test <,<=,==,>= • Slow	• Test only one variable • Primitive: Test only byte, short, char, int • Covering: String, Byte, Short, Character, Integer • Test only equality • Faster

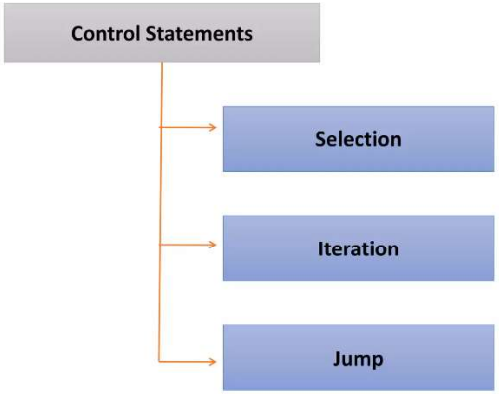
- false && ... - it is not necessary to know what the right hand side is, the result must be false
- true || ... - it is not necessary to know what the right hand side is, the result must be true

HNDIT2012
Fundamentals of Programming

Java Control Statements

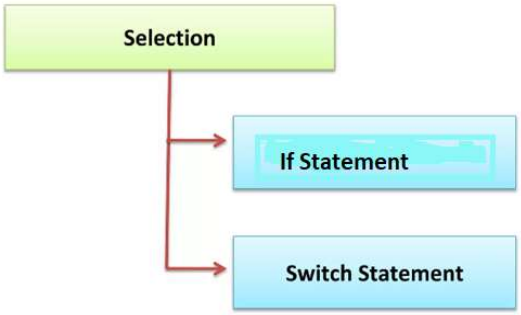
Types of Control Flow Statements in Java

- Selection statements
- Iteration statements
- Jump statements

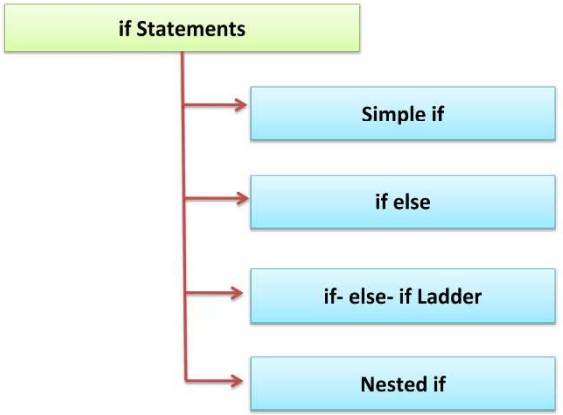


Selection Statements

- Selection Statements are also called Decision Making Statements.



if Statements



Simple if

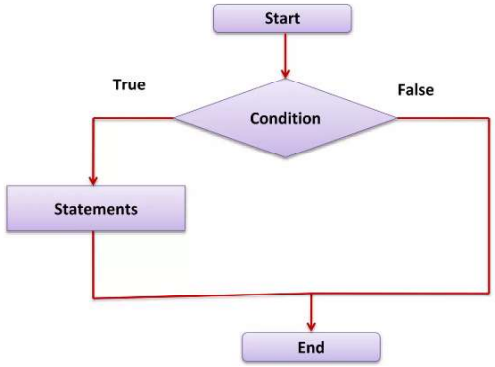
Syntax :

```
if (condition)
{
   statement1;
}
```

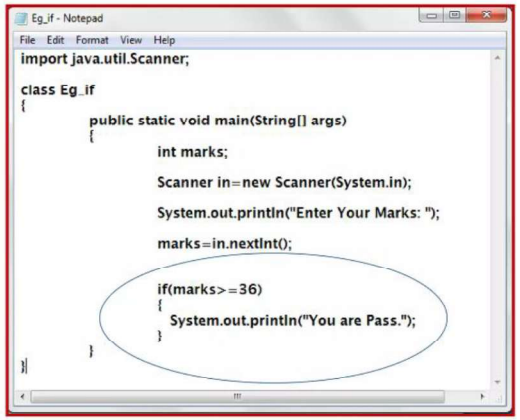
Purpose: The statements will be evaluated if the value of the condition is true.

Simple if

Flow Chart:



Example



if else

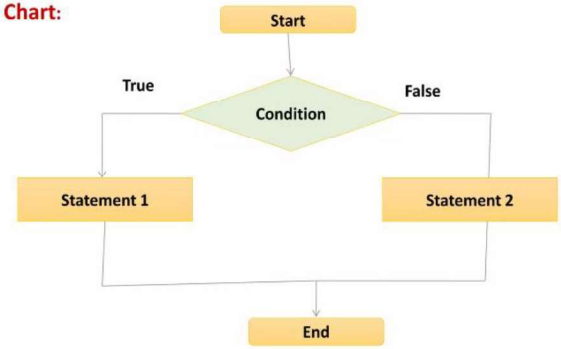
Syntax :

```
if (condition)
{
   statement1;
}
else
{
   statement2;
}
```

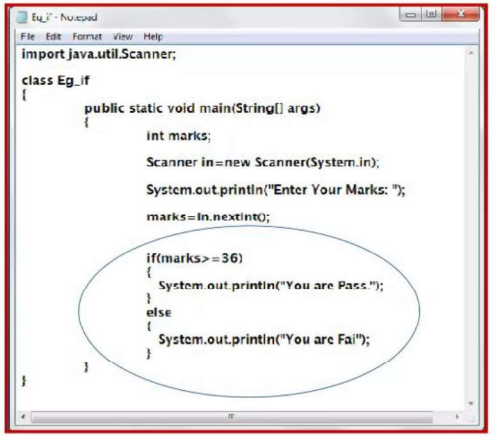
Purpose: The statement 1 is evaluated if the value of the condition is true otherwise statement 2 is true.

if else

Flow Chart:



Example



```
import java.util.Scanner;

class Eg_if
{
        public static void main(String[] args)
        {
                int marks;

                Scanner in=new Scanner(System.in);

                System.out.println("Enter Your Marks: ");

                marks=in.nextInt();

                if(marks>=36)
                {
                  System.out.println("You are Pass.");
                }
                else
                {
                  System.out.println("You are Fai");
                }
        }
}
```


If-else-if Ladder

Syntax :

```
if(condition)
 statements;
else if(condition)
 statements;
else if(condition)
 statements;
...
...
else
 statements;
```


Example

```
public class IfElseLadderExample {
    public static void main(String args[]) {
        int age = 55;
        if (age >= 60) {
            System.out.println("Person is double-vaccinated");
        } else if (age >= 50 && age < 60) {
            System.out.println("Person is vaccinated with single-dose only");
        } else if (age >= 40 && age < 50) {
            System.out.println("Person is not vaccinated");
        } else {
            System.out.println("Person is not eligible for vaccine ");
        }
    }
}
```

Output:

```
Person is vaccinated with single-dose only
```

Nested if

- A nested if is an if statement that is the target of another if or else.
- Nested ifs are very common in programming.

Syntax :

```
if(condition)
{
   if(condition)
      statements....
   else
      statements....
}
else
{
    if(condition)
      statements....
   else
      statements....
}
```


13

Example

```
1  import java.util.Scanner;
2  class MaxValue
3  {
4          public static void main(String args[])
5          {
6                  int  a,b,c;
7                  int max=0;
8                  Scanner s = new Scanner(System.in);
9                  System.out.println("Enter value for a : ");
10                 a=s.nextInt();
11                 System.out.println("Enter value for b : ");
12                 b=s.nextInt();
13                 System.out.println("Enter value for c : ");
14                 c=s.nextInt();
15                 if (a>b)
16                 {
17                         if(a>c)
18                            max=a;
19                         else        //This else is associate with this if(a>c)
20                            max=c;
21                 }
22                 else               //This else is associate with this if(a>b)
23                 {
24                         if(b>c)
25                            max=b;
26                         else        //This else is associate with this if(b>c)
27                            max=c;
28                 }
29                 System.out.println("\n max value = " +max);
30         }
31 }
```


switch

Syntax :

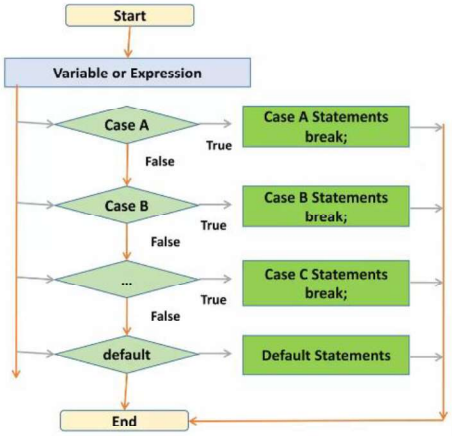
```
switch (expression)
{
case value 1 :
statement 1 ; break;
case value 2 :
statement 2 ; break;
...
...
case value N :
statement N ; break;
default :
statements ; break;
}
```

Purpose: The statements N will be evaluated if the value of the logical expression is true.

15

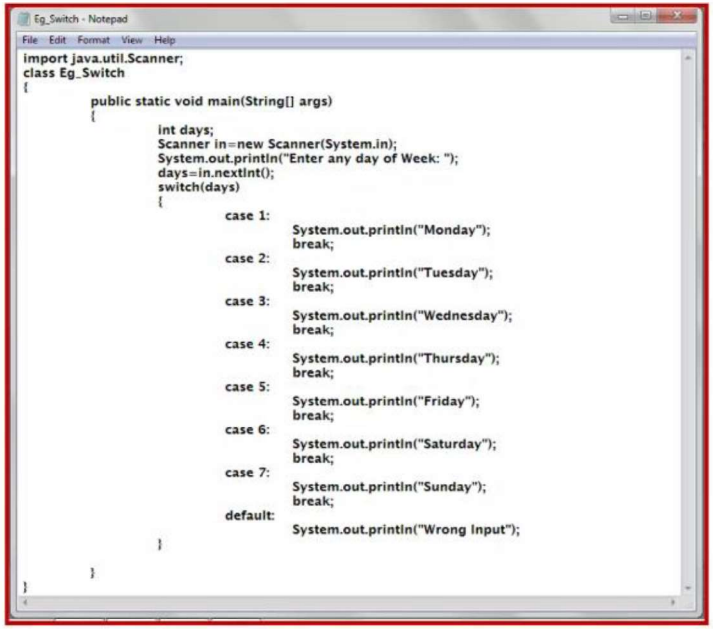
switch

Flow Chart:



16

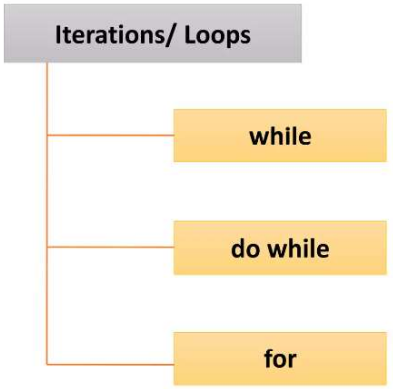
Example



Iteration Statements

Each loop has four types of statements :

- ✓ Initialization
- ✓ Condition checking
- ✓ Execution
- ✓ Increment / Decrement



while

Syntax:

```
initialization
while(final value)
{
 statements;
 increment/decrement;
}
```

```
m=1
while(m<=20)
{
 System.out.println(m);
 m=m+1;
}
```

Purpose: To evaluate the statements from initial value to final value with given increment/decrement.

Example

- **print values from 1 to 10**

```
class while1
{
    public static void main(String args[])
    {
        int i=1;
        while(i<=10)
        {
            System.out.println("\n" + i);
            i++;
        }
    }
}
```

Output :

1
2
3
4
5
6
7
8
9
10

do while

Syntax:

```
initialization
do
{
 statements;
 increment/decrement;
}
 while(final value);
```

```
m=1
do
{
 System.out.println(m);
 m=m+1;
}
while(m==20);
```

Purpose: To evaluate the statements from initial value to final value with given increment/decrement.

Example

```
class dowhile1
{
    public static void main(String args[])
    {
        int i = 1;
        int sum = 0;
        do
        {
            sum = sum + i;
            i++;
        }while (i<=10);
        System.out.println("\n\n\tThe sum of 1 to 10 is .. " + sum);
    }
}
```

Output :

```
The sum of 1 to 10 is .. 55
```


22

for

Syntax:

```
for(initialization;final value;increment/decrement)
{
 statements;
}
```

```
for(m=1;m<=20;m=m+1)
{
 System.out.println(m);
}
```

Purpose: To evaluate the statements from initial value to final value with given increment/decrement.

Example

```
class for1
{
    public static void main(String args[])
    {
        int i;
        for (i=0;i<5;i++)
        {
            System.out.println("\nExample of for loop ");
        }
    }
```

Output :

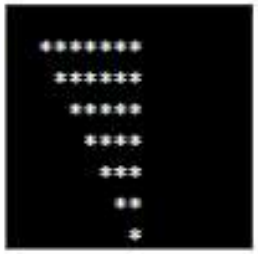
```
Example of for loop
Example of for loop
Example of for loop
Example of for loop
Example of for loop
```


24

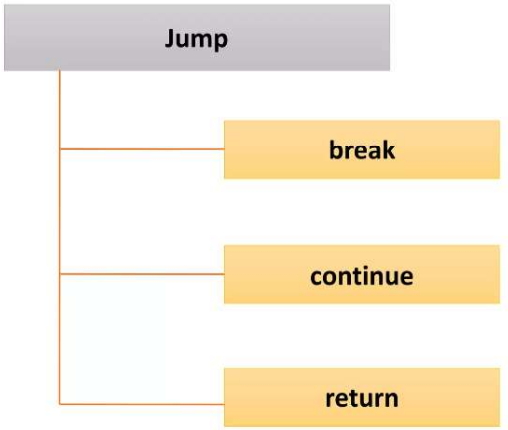
. Right Down Mirror Star Pattern

```
public class RightDownMirrorPattern
{
public static void main(String args[])
{
int row=7;
for (int i= row; i>= 1; i--)
{
for (int j=row; j>i;j--)
{
System.out.print(" ");
}
for (int k=1;k<=i;k++)
{
System.out.print("*");
}
System.out.println("");
}
}
}
```

Output:



Jump Statements



The break statement

- ✓ This statement is used to jump out of a loop.
- ✓ Break statement was previously used in switch – case statements.
- ✓ On encountering a break statement within a loop, the execution continues with the next statement outside the loop.
- ✓ The remaining statements which are after the break and within the loop are skipped.
- ✓ Break statement can also be used with the label of a statement.
- ✓ A statement can be labeled as follows.

statementName : SomeJavaStatement

- ✓ When we use break statement along with label as,

break statementName;

Example

```
class break1
{
    public static void main(String args[])
    {
        int i = 1;
        while (i<=10)
        {
            System.out.println("\n" + i);
            i++;
            if (i==5)
            {
                break;
            }
        }
    }
}
```

Output :

1
2
3
4

continue Statement

- ✓ This statement is used only within looping statements.
- ✓ When the continue statement is encountered, the next iteration starts.
- ✓ The remaining statements in the loop are skipped. The execution starts from the top of loop again.

Example

```
class continue1
{
    public static void main(String args[])
    {
        for (int i=1; i<10 ; i++)
        {
            if (i%2 == 0)
                continue;

            System.out.println("\n" + i);
        }
    }
}
```

```
Output :
1
3
5
7
9
```


The return Statement

- ✓ The last control statement is return. The return statement is used to explicitly return from a method.
- ✓ That is, it causes program control to transfer back to the caller of the method.
- ✓ The return statement immediately terminates the method in which it is executed.

Example

```
class Return1
{
    public static void main(String args[])
    {
        boolean t = true;
        System.out.println("Before the return.");
        if(t)
            return;     // return to caller
        System.out.println("This won't execute.");
    }
}
```

```
Output :
Before the return.
```

Recap…

Control Structure	Purpose	Syntax
`if ... else`	Used to write a decision with *conditions* that select the alternative to be executed. Executes the first (second) alternative if the *condition* is true (false).	`if (condition) { ... } else { ... }`
`switch`	Used to write a decision with scalar values (integers, characters) that select the alternative to be executed. Executes the *statements* following the *label* that is the *selector* value. Execution falls through to the next `case` if there is no `return` or `break`. Executes the statements following `default` if the *selector* value does not match any *label*.	`switch (selector) { case label : statements; break; case label : statements; break; ... default : statements; }`
`while`	Used to write a loop that specifies the repetition *condition* in the loop header. The *condition* is tested before each iteration of the loop and, if it is true, the loop body executes; otherwise, the loop is exited.	`while (condition) { ... }`
`for`	Used to write a loop that specifies the *initialization*, repetition *condition*, and *update* steps in the loop header. The *initialization* statements execute before loop repetition begins; the *condition* is tested before each iteration of the loop and, if it is true, the loop body executes; otherwise, the loop is exited. The *update* statements execute after each iteration.	`for (initialization; condition; update) { ... }`
`do ... while`	Used to write a loop that specifies the repetition *condition* after the loop body. The *condition* is tested after each iteration of the loop and, if it is true, the loop body is repeated; otherwise, the loop is exited. The loop body always executes at least one time.	`do { ... } while (condition) ;`

HNDIT2022-Software Development

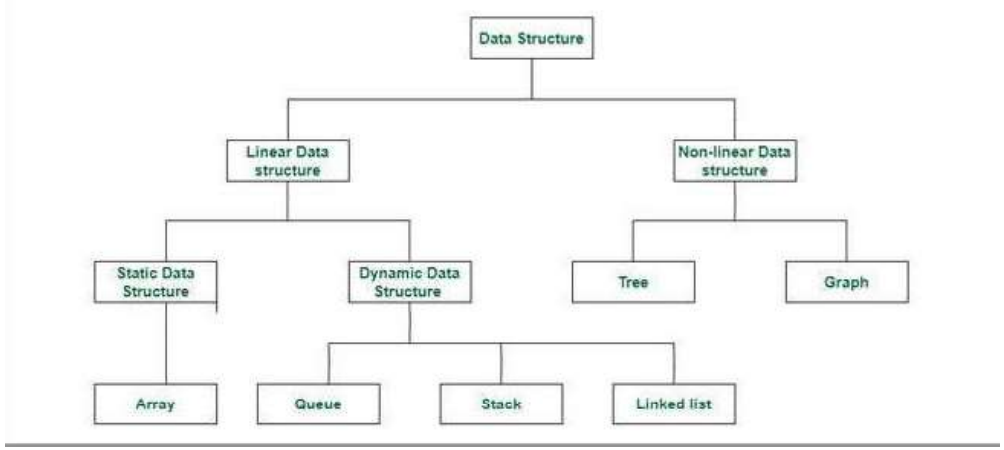
Week 6: Data Structures & Data abstraction

What is a Data Structure?

A data structure is a storage that is used to store and organize data so that it can be used efficiently.

- A data structure is not only used for organizing the data.
- It is also used for processing, retrieving, and storing data.
- Data types are often confused as a type of data structures, but it is not precisely correct even though they are referred to as Abstract Data Types. Data types represent the nature of the data while data structures are just a collection of similar or different data types in one.

Classification of Data Structure



Linear Data Structure

- Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, is called a linear data structure.
  Examples of linear data structures are array, stack, queue, linked list, etc.

- **Static data structure:** Static data structure has a fixed memory size. It is easier to access the elements in a static data structure.
  An example of this data structure is an array.
- **Dynamic data structure:** In dynamic data structure, the size is not fixed. It can be randomly updated during the runtime which may be considered efficient concerning the memory (space) complexity of the code.
  Examples of this data structure are queue, stack, etc.

Non-linear Data Structure

- Data structures where data elements are not placed sequentially or linearly are called non-linear data structures. In a non-linear data structure, we can't traverse all the elements in a single run only.
  Examples of non-linear data structures are trees and graphs.

Major Operations

- **Searching:** We can search for any element in a data structure.
- **Sorting:** We can sort the elements of a data structure either in an ascending or descending order.
- **Insertion:** We can also insert the new element in a data structure.
- **Updating:** We can also update the element, i.e., we can replace the element with another element.
- **Deletion:** We can also perform the delete operation to remove the element from the data structure.

Advantages of Data Structures

The following are the advantages of a data structure:

- **Efficiency:** If the choice of a data structure for implementing a particular ADT is proper, it makes the program very efficient in terms of time and space.
- **Reusability:** The data structure provides reusability means that multiple client programs can use the data structure.
- **Abstraction:** The data structure specified by an ADT also provides the level of abstraction. The client cannot see the internal working of the data structure, so it does not have to worry about the implementation part. The client can only see the interface.

Abstract Data Type

- An abstract data type is an abstraction of a data structure that provides only the interface to which the data structure must adhere. The interface does not give any specific details about something should be implemented or in what programming language.

Collections

- A collection is a concept applicable to abstract data types, and does not prescribe a specific implementation as a concrete data structure, though often there is a conventional choice.
- Examples of collections include lists, sets, multisets, trees and graphs.

Many collections define a particular linear ordering, with access to one or both ends. The actual data structure implementing such a collection need not be linear—for example, a priority queue is often implemented as a heap, which is a kind of tree. Important linear collections include:

- lists;
- stacks;
- queues;
- priority queues;
- double-ended queues;
- double-ended priority queues.

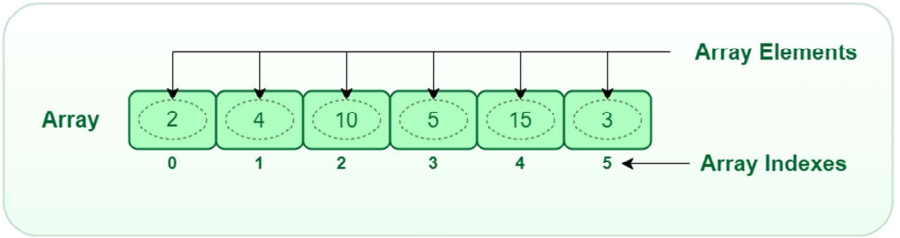
Arrays

- Arrays are defined as the collection of similar types of data items stored at contiguous memory locations.
- It is one of the simplest data structures where each data element can be randomly accessed by using its index number.

Properties of Array

- Each element in an array is of the same data type and carries the same size that is 4 bytes.
- Elements in the array are stored at contiguous memory locations from which the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Representation of an Array



Basic Operations in the Arrays

- **Traverse** − print all the array elements one by one.
- **Insertion** − Adds an element at the given index.
- **Deletion** − Deletes an element at the given index.
- **Search** − Searches an element using the given index or by the value.
- **Update** − Updates an element at the given index.
- **Display** − Displays the contents of the array.

Insertion Operation in Arrays

- Algorithm
  1. Start
  2. Create an Array of a desired datatype and size.
  3. Initialize a variable ‘i’ as 0.
  4. Enter the element at ith index of the array.
  5. Increment i by 1.
  6. Repeat Steps 4 & 5 until the end of the array.
  7. Stop

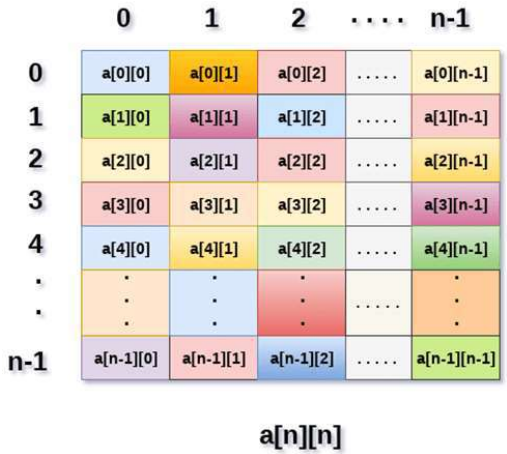
Deletion Operation

- Algorithm-
  1. Start
  2. Set J = K
  3. Repeat steps 4 and 5 while J < N
  4. Set LA[J] = LA[J + 1]
  5. Set J = J+1
  6. Set N = N-1
  7. Stop

2D Array

- 2D array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns.

int arr[max_rows][max_columns];



Advantages of Array

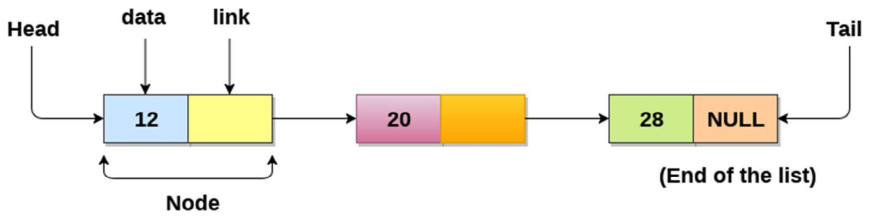
- Array provides the single name for the group of variables of the same type. Therefore, it is easy to remember the name of all the elements of an array.
- Traversing an array is a very simple process; we just need to increment the base address of the array in order to visit each element one by one.
- Any element in the array can be directly accessed by using the index.

Disadvantages of Array

- Array is homogenous. It means that the elements with similar data type can be stored in it.
- In array, there is static memory allocation that is size of an array cannot be altered.
- There will be wastage of memory if we store less number of elements than the declared size.

Linked Lists

- Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.



As per the above illustration, following are the important points to be considered.
Linked List contains a link element called first (head).
Each link carries a data field(s) and a link field called next.
Each link is linked with its next link using its next link.
Last link carries a link as null to mark the end of the list.

Uses of Linked List

- The list is not required to be contiguously present in the memory. The node can reside any where in the memory and linked together to make a list. This achieves optimized utilization of space.
- List size is limited to the memory size and doesn't need to be declared in advance.
- Empty node can not be present in the linked list.
- We can store values of primitive types or objects in the singly linked list.

Why use linked list over array?

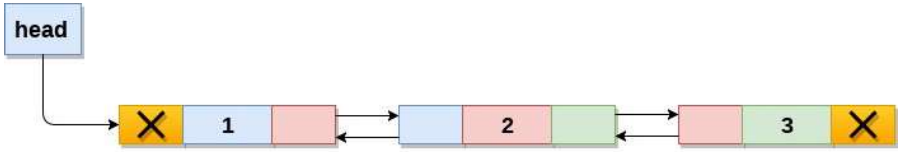
Till now, we were using array data structure to organize the group of elements that are to be stored individually in the memory. However, Array has several advantages and disadvantages which must be known in order to decide the data structure which will be used throughout the program.

- Array contains following limitations:
- The size of array must be known in advance before using it in the program.
- Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
- All the elements in the array need to be contiguously stored in the memory. Inserting any element in the array needs shifting of all its predecessors.

Linked list is the data structure which can overcome all the limitations of an array. Using linked list is useful because,

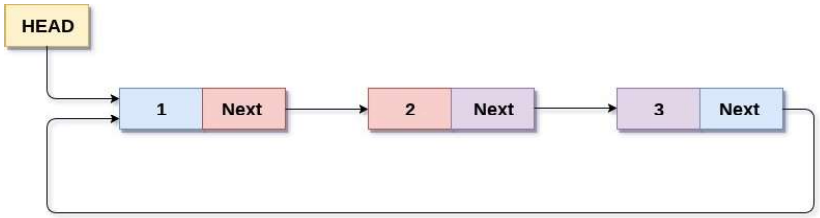
- It allocates the memory dynamically. All the nodes of linked list are non-contiguously stored in the memory and linked together with the help of pointers.
- Sizing is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and limited to the available memory space.

Doubly Linked List



Doubly Linked List

Circular Singly Linked List



Circular Singly Linked List

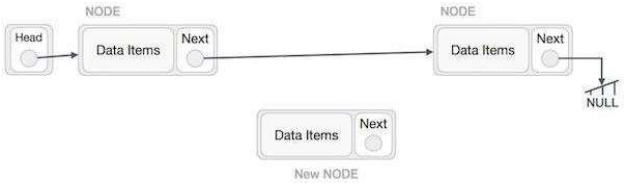
Basic Operations in the Linked Lists

The basic operations in the linked lists are insertion, deletion, searching, display, and deleting an element at a given key. These operations are performed on Singly Linked Lists as given below –

- **Insertion** – Adds an element at the beginning of the list.
- **Deletion** – Deletes an element at the beginning of the list.
- **Display** – Displays the complete list.
- **Search** – Searches an element using the given key.
- **Delete** – Deletes an element using the given key.

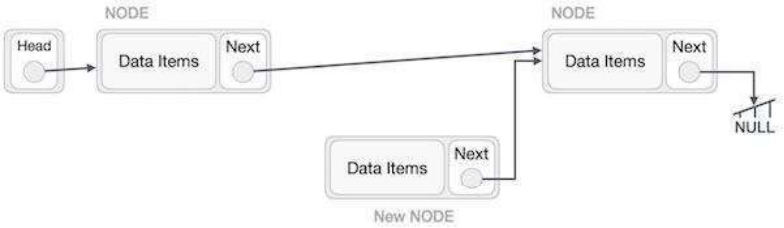
Insertion Operation

- Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.



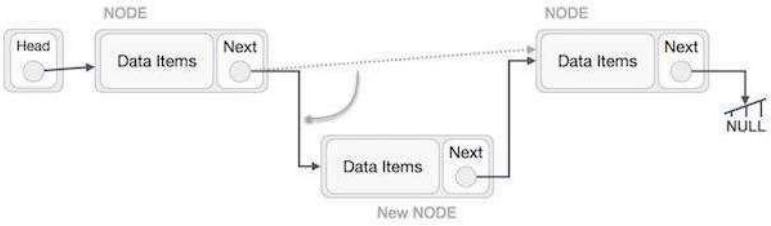
- Imagine that we are inserting a node B (NewNode), between A (LeftNode) and C (RightNode). Then point B.next to C –

  NewNode.next –> RightNode;



- Now, the next node at the left should point to the new node.

  LeftNode.next –> NewNode;



Insertion at Beginning

1. START
2. Create a node to store the data
3. Check if the list is empty
4. If the list is empty, add the data to the node and assign the head pointer to it.
5. If the list is not empty, add the data to a node and link to the current head. Assign the head to the newly added node.
6. END

Insertion at Ending

1. START
2. Create a new node and assign the data
3. Find the last node
4. Point the last node to new node
5. END

Insertion at a Given Position

1. START
2. Create a new node and assign data to it
3. Iterate until the node at position is found
4. Point first to new first node
5. END

Deletion Operation

- Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.

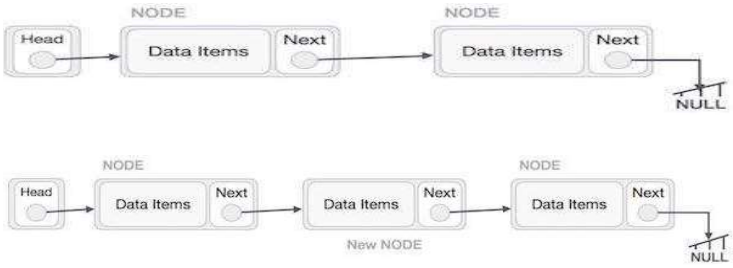
- The left (previous) node of the target node now should point to the next node of the target node −

LeftNode.next –> TargetNode.next;

This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

TargetNode.next –> NULL;

We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.



Similar steps should be taken if the node is being inserted at the beginning of the list. While inserting it at the end, the second last node of the list should point to the new node and the new node will point to NULL.

Deletion at Beginning

In this deletion operation of the linked, we are deleting an element from the beginning of the list. For this, we point the head to the second node.

Algorithm-

1. START
2. Assign the head pointer to the next node in the list
3. END

Deletion at Ending

In this deletion operation of the linked, we are deleting an element from the ending of the list.

Algorithm-

1. START
2. Iterate until you find the second last element in the list.
3. Assign NULL to the second last element in the list.
4. END

Deletion at a Given Position

In this deletion operation of the linked, we are deleting an element at any position of the list.

Algorithm-

1. START
2. Iterate until find the current node at position in the list
3. Assign the adjacent node of current node in the list to its previous node.
4. END

Tuples

- Tuples are used to store multiple items in a single variable.
- Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.
- A tuple is a collection which is ordered and **unchangeable**.
- Tuples are written with round brackets.

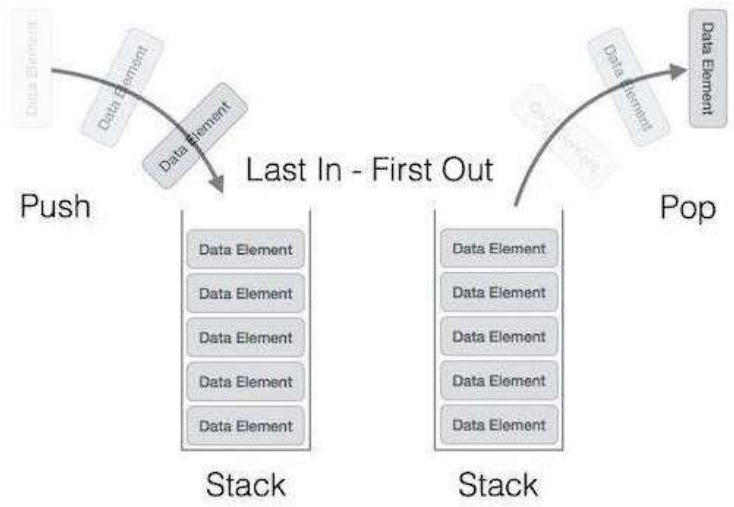
```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Stack

A stack is an Abstract Data Type (ADT), that is popularly used in most programming languages. It is named stack because it has the similar operations as the real-world stacks, for example – a pack of cards or a pile of plates, etc. The stack follows the LIFO (Last in - First out) structure where the last element inserted would be the first element deleted.

Stack Representation

- A Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.
- A stack can be implemented by means of Array, Structure, Pointer, and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.



Basic Operations on Stacks

- Stack operations usually are performed for initialization, usage and, de-initialization of the stack ADT.
- The most fundamental operations in the stack ADT include: push(), pop(), peek(), isFull(), isEmpty(). These are all built-in operations to carry out data manipulation and to check the status of the stack.
- Stack uses pointers that always point to the topmost element within the stack, hence called as the **top** pointer.

Insertion: push()

- push() is an operation that inserts elements into the stack. The following is an algorithm that describes the push() operation in a simpler way.

Algorithm-

1 − Checks if the stack is full.

2 − If the stack is full, produces an error and exit.

3 − If the stack is not full, increments top to point next empty space.

4 − Adds data element to the stack location, where top is pointing.

5 − Returns success.

Deletion: pop()

pop() is a data manipulation operation which removes elements from the stack. The following pseudo code describes the pop() operation in a simpler way.

Algorithm-

1 − Checks if the stack is empty.

2 − If the stack is empty, produces an error and exit.

3 − If the stack is not empty, accesses the data element at which top is pointing.

4 − Decreases the value of top by 1.

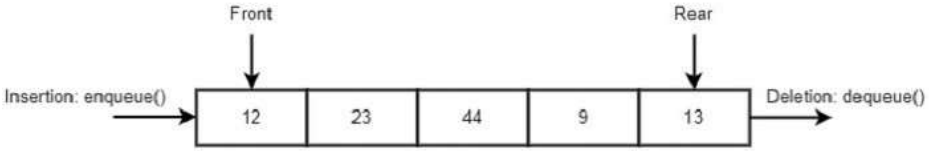
5 − Returns success.

Queue

- Queue, like Stack, is also an abstract data structure. The thing that makes queue different from stack is that a queue is open at both its ends. Hence, it follows FIFO (First-In-First-Out) structure, i.e. the data item inserted first will also be accessed first. The data is inserted into the queue through one end and deleted from it using the other end.

A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops.

Representation of Queues



Queue: FIFO Operation

Basic Operations

- The most fundamental operations in the queue ADT include: enqueue(), dequeue(), peek(), isFull(), isEmpty(). These are all built-in operations to carry out data manipulation and to check the status of the queue.
- Queue uses two pointers – **front** and **rear**. The front pointer accesses the data from the front end (helping in enqueueing) while the rear pointer accesses data from the rear end (helping in dequeuing).

Insertion Operation: enqueue()

- The enqueue() is a data manipulation operation that is used to insert elements into the stack. The following algorithm describes the enqueue() operation in a simpler way.

Algorithm

1 – START
2 – Check if the queue is full.
3 – If the queue is full, produce overflow error and exit.
4 – If the queue is not full, increment rear pointer to point the next empty space.
5 – Add data element to the queue location, where the rear is pointing.
6 – return success.
7 – END

Deletion Operation: dequeue()

The dequeue() is a data manipulation operation that is used to remove elements from the stack. The following algorithm describes the dequeue() operation in a simpler way.

Algorithm-

1 – START
2 – Check if the queue is empty.
3 – If the queue is empty, produce underflow error and exit.
4 – If the queue is not empty, access the data where front is pointing.
5 – Increment front pointer to point to the next available data element.
6 – Return success.
7 – END

Data Abstraction

- Data abstraction is the reduction of a particular body of data to a simplified representation of the whole. Abstraction, in general, is the process of removing characteristics from something to reduce it to a set of essential elements.

Thank you..

HNDIT2012 Fundamentals of Computer Programming

Introduction to IDE

Software Versioning

Software versioning

Software versioning is the process of assigning either unique version names or unique version numbers to unique states of computer software.

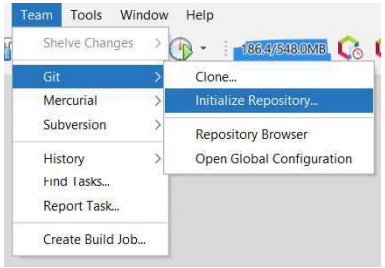
- Within a given version number category, these numbers are generally assigned in increasing order and correspond to new developments in the software.

Versioning with NetBeans

Apache NetBeans IDE provides support for the Git, Mercurial and Subversion version control systems, which are distributed version control and source code management systems

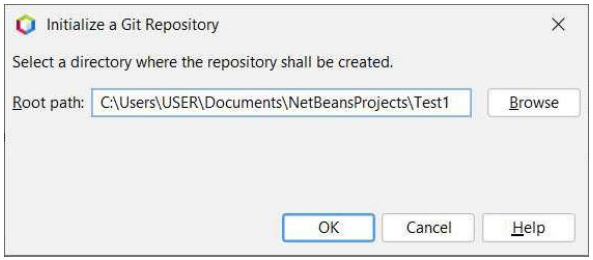
Git versioning with NetBeans

- Initialize a repository



Initialize a repository

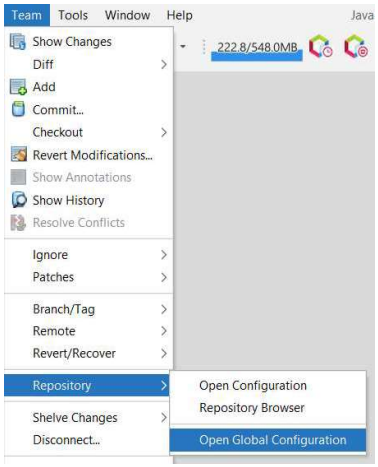
Repository allows developer to save versions of the code, which can access when needed.



A .git subfolder is created in current working directory, which all the project snapshots are stored. This will also create a new main branch.

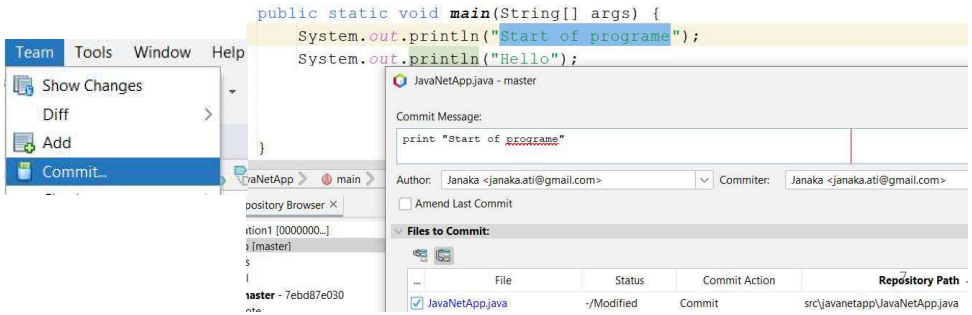
C:\Users\USER\Documents\NetBeansProjects\Test1\.git

Global configuration



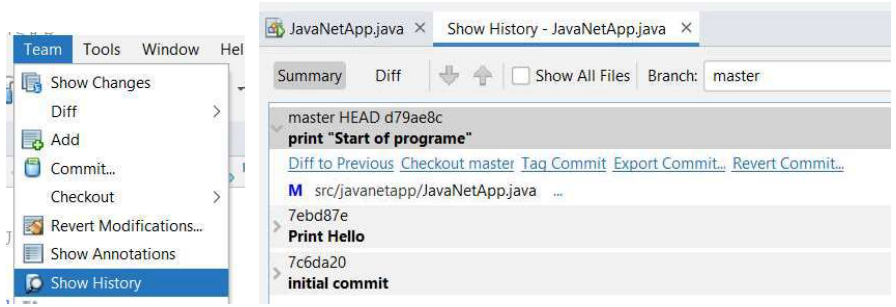
Commit

- Commit means snapshots or milestones along the timeline of a Git project.
- So, developer can do time to time changes in project and commit with a suitable message.



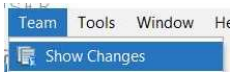
View Commit history

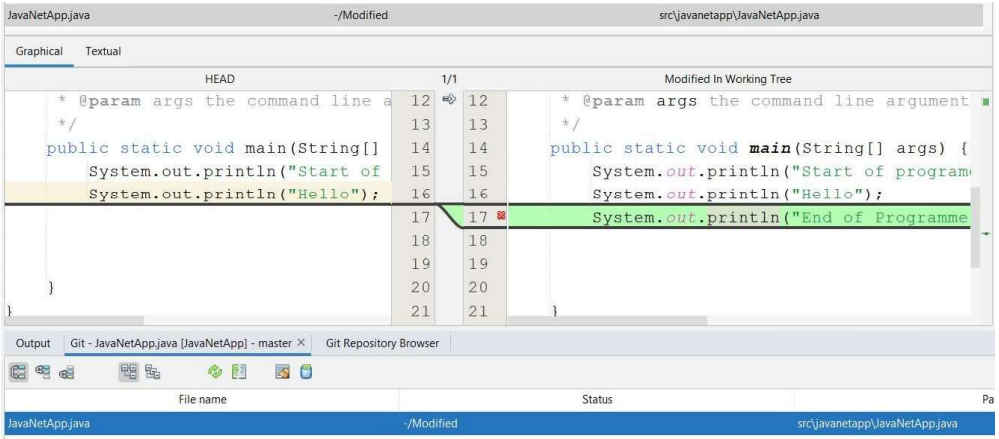
- Display the commit history



Show changes

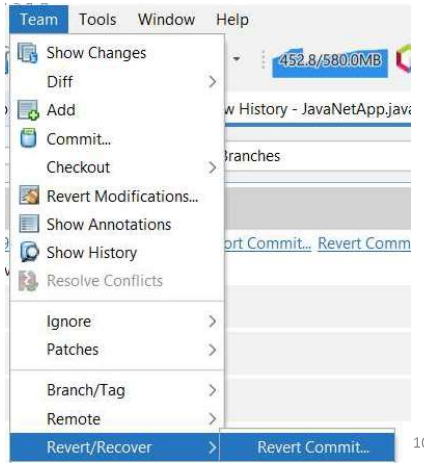
- Display the changes in project





Revert commit

- Reverse back to previous commit



HNDIT2012 Fundamentals of Computer Programming

Error Handling

Program Errors

- Syntax errors (compile time)
  - Caused by an incorrectly written code such as misspelled keyword or variable, and incomplete code.
- Runtime errors
  - Occurs when a statement attempts an operation that is impossible to carry out.
  - Ex: division by zero.
- Logical errors
  - Occurs when code complies and runs without generating an error, but the result produced is incorrect.

Run time errors

- Occurs at program run time
- Condition that changes the normal flow of execution
  - Program run out of memory
  - File does not exist in the given path
  - Network connections are dropped
  - OS limitations

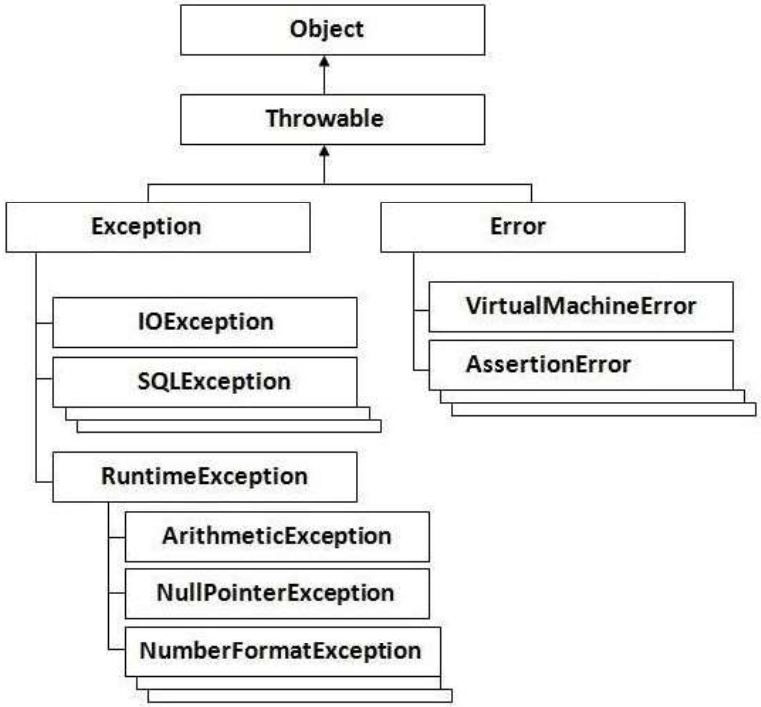
Exception

- Unexpected behavior of program
- An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

exception object

- When an error occurs within a method, the method creates an object and hands it off to the runtime system.
- The object, called an *exception object.*
- Contains information about the error, including its type and the state of the program when the error occurred.
- Creating an exception object and handing it to the runtime system is called *throwing an exception*.

Hierarchy of Java Exception classes



Types of Java Exceptions

- Checked exceptions
  - ArithmeticException
  - ClassCastException
  - NullPointerException
  - ArrayIndexOutOfBoundsException
  - NegativeArraySizeException
  - ArrayStoreException
  - IllegalThreadStateException
  - SecurityException, etc.
- Unchecked
  - Runtime exceptions
  - Errors

Types of Exceptions

- Checked exceptions
  - Occurs at the compile time
  - Also called as compile time exceptions.
  - Cannot be ignored at the time of compilation by programmer.
  - All exceptions are checked exceptions, other than Error, RuntimeException, and their subclasses

- IOException
- SQLException
- ClassNotFoundException
- IllegalAccessException

Types of Exceptions

- Runtime exceptions
  - At the time of execution.
  - Never checked
  - These include programming bugs, such as logic errors or improper use of an API.

- ArithmeticException
- NullPointerException
- ArrayIndexOutOfBoundsException
- InvalidArgumentException
- NumberFormatException

Types of Exceptions

- Errors
  - Not exceptions at all, but problems that arise beyond the control of the user or the programmer.
  - Errors are typically ignored in your code because you can't rarely do anything about an error.
  - For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

OutOfMemoryError
VirtualMachineError
AssertionError

Exception handling

- **Exception handling in java** is one of the powerful *mechanism to handle the runtime errors*

try catch block

```
try
    {
    //Protected code
    }
catch(ExceptionName e1)
    {
    //Catch block
    }
```

try catch block

- Example 1

```
public class Main
{
   public static void main(String[] args) {
      try{
           int a=4, b=0;
           int c=a/b;
           System.out.println(c);
           }
      catch(ArithmeticException E)
           {
           System.out.print("Error"+ E.toString());
          }
   }
}
```

Multiple catch Blocks

```
try
        { //Protected code
        }
catch(ExceptionType1 e1)
        { //Catch block
        }
catch(ExceptionType2 e2)
        { //Catch block
        }
catch(ExceptionType3 e3)
        { //Catch block
        }
```

try catch block

- Example 2

```
public class Main
{
   public static void main(String[] args) {
      try{
      int a=4;
      int b=0;
      int c=a/b;
      System.out.println(c);
      }
      catch(ArithmeticException E)
      {
      System.out.print("Error"+ E.toString());
      }
   }
}
```

finally Block

- A finally block of code always executes, whether or not an exception has occurred.

try catch with finally

- Example 3

```
public class ExcepTest
{   public static void main(String[] args)
    {
      try{
         int a=4;
         int b=0;
         int c=a/b;
         System.out.println(c);
          }
      catch(ArithmeticException E)
       {        System.out.print("Error"+ E.toString());         }
      finally
       {        System.out.print("End");        }
   }
}
```

Throwable

- Superclass of all errors and exceptions
- All exceptions and errors extend from a common java.lang.Throwable parent class

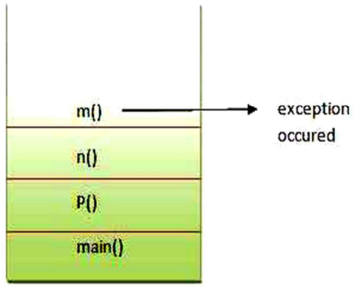
Nesting try blocks

```
public class Demo
  {
  public static void main(String args[])
    {
    try
      {
      try
        {
        System.out.println("going to divide");
        int b =39/0;
        }
      catch(ArithmeticException e)
        {        System.out.println(e);         }
      System.out.println("other statement");
      }
    catch(Exception e){System.out.println("handeled");}
    System.out.println("normal flow..");
    }
  }
```

Exception Propagation

```
class Demo
{
  void m(){   int data=50/0;  }
  void n(){    m();   }
  void p()
       {
     try{
         n();
        }
     catch(Exception e)
        {System.out.println("exception handled");}
     }
 public static void main(String args[])
       {
     Demo obj=new Demo();
     obj.p();
     System.out.println("normal flow...");
    }
}
```

An exception is first thrown from the top of the stack is not caught, it drops down to the previous method in the call stack.



throw

- The Java throw keyword is used to explicitly throw an exception.
- We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

```
throw new ArithmeticException("not valid");
```

- Refer **Lab8**

throws

- Keyword in the signature of method to indicate that method might throw one of the listed type exceptions
- Use throws keyword to delegate the responsibility of exception handling to the caller (method or JVM), then caller method is responsible to handle that exception.
- throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.
- Only checked exceptions are required to be thrown using the throws keyword

```
class TestVar
{
   static void fun() throws IllegalAccessException
   {
      System.out.println("Inside fun(). ");
      throw new IllegalAccessException("demo");
   }
   public static void main(String args[])
   {
      try
      {          fun();       }
      catch(Exception e)
      {
         System.out.println("Caught in main"+e.toString());
      }
   }
}
```

Fundamentals of Programming

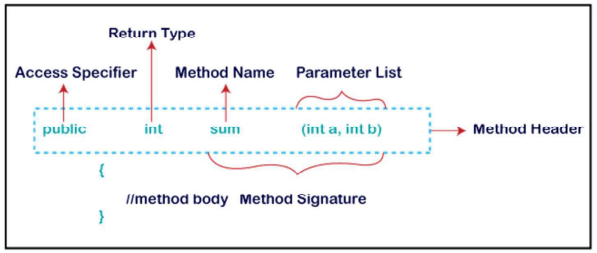
Methods in Java
Lecture 09

What is a method ?

- A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.
- They are also known as **functions**.
- The method is executed only when we call or invoke it.
- Why use methods?
  - To **reuse code**: define the code once, and use it many times.
  - Provides **easy modification** of the code, just by adding or removing a chunk of code.

Method Declaration

- A method must be declared within a class.
- The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments.
- It has six components that are known as **method header**, as we have shown in the following figure.



Method Declaration - Access Specifier

- Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:
  - **Public:** The method is accessible by all classes when we use public specifier in our application.
  - **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
  - **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
  - **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Method Declaration

- **Return Type:** Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.
- **Method Name:** It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.
- **Parameter List:** It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.
- **Method Body:** It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Types of Methods

- There are two types of methods in Java:
  1. User defined Methods
     - Written by the programmer according to the requirements.
  2. Predefined Method
     - Method that is already defined in the Java class libraries.
     - Also known as the **standard library method** or **built-in method**.
     - Eg: **length(), equals(), compareTo(), sqrt(),** etc.

User defined Methods (Example 01)

- Write a method to print the string "I want to draw a circle.".

```
class Circle{

        void display(){
            System.out.println("I want to draw a circle.");
        }
}
```

Call or invoke a user defined method

```
class Circle{

        static void display(){
            System.out.println("I want to draw a circle.");
        }

        public static void main(String[] args) {
            display();
        }
}
```

User defined Methods (Example 02)

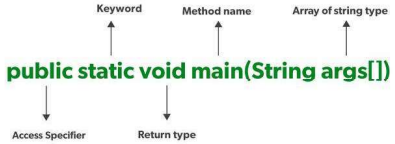
- Write a method to get two integers and return the total.

```
class Calculator{

        static int addNumbers(int num1,int num2){
            int total = num1 + num2;
            return total;
        }

        public static void main(String[] args) {
            int number1 = 10, number2 = 20;

            int answer = addNumbers(number1,number2);
            System.out.println("The total is" + answer);

            System.out.println("The total is" + addNumbers(number1,number2));
        }
}
```

Predefined Method

```
public class FindMax{
    public static void main(String[] args){
        System.out.print("The maximum number is: " + Math.max(9,7));
    }
}
```

- In the code above three predefined methods main(), print(), and max() has used directly without declaration.
- print() method is defined in the **java.io.PrintStream** class. It prints the statement that we write inside the method. For example, print("Java"), it prints Java on the console.
- The max() method is a method of the **Math** class that returns the greater of two numbers.
- The main() is the starting point for JVM to start execution of a Java program. Without the main() method, JVM will not execute the program.



Static Method vs Instance Method

Static Method

- Method that belongs to a class rather than an instance of a class.
- Can call it without creating an object.
- It is invoked by using the class name.

Instance Method

- Method belongs to a instance of a class.
- Must create a object of the class to invoke it.
- It is invoked by using the object name.

Static Method vs Instance Method

Static Method

```
public class Circle {
    public static void main(String[] args){
        calArea(7);
    }

    static void calArea(int radius)
    {
        double area = Math.PI*radius*radius;
        System.out.println(area);
    }
}
```

Instance Method

```
public class Circle {
    public static void main(String[] args){
        Circle c1 = new Circle();
        c1.calArea(7);
    }

    void calArea(int radius)
    {
        double area = Math.PI*radius*radius;
        System.out.printf("%.02f", area);
    }
}
```

HNDIT2012
Fundamentals of Computer Programming

Week 10 – Access Modifiers, Classes and Objects

Access Modifiers

Lesson Introduction

- This lesson provides the knowledge about Access Modifiers that specify the accessibility or scope of a field, method, constructor, or class. It also throws in the knowledge of Classes and Objects, the basic concepts of Object Oriented Programming that revolve around real-life entities.

• Modifiers

- Modifiers are Language Keywords that modify the definition of a Class, Method or a Variable
- Access to variables and methods in Java classes is accomplished through Access Modifiers
- Access modifiers define varying levels of access between class members and the outside world (other objects)

Access modifiers

- Access modifiers are declared immediately before the type of a member variable or the return type of a method.

- There are four access modifiers :
  - Default
  - Public
  - Protected
  - Private

1. Default access modifier

- The Default access modifier is applied by default in the absence of an access modifier.

- Specifies that only classes in the same package (groups of related classes and interfaces) can have access to class's variables and methods

```
long length;  void getlength()
{
return length;
}
```

- Notice that neither the member variable nor the method, supply an access modifier.

- So they take on the default access modifier implicitly

2. public access modifier

- Specifies that class variables and methods are accessible to anyone, both inside and outside the class

- This means that public class members have global visibility and can be accessed by any other objects.

```
public int count;
public boolean isActive;
```


3. protected access modifier

- Specifies that class members are accessible only to
- methods in that class and subclasses of that class.

- This means that protected class members have visibility
- limited to subclasses.

```
protected char middleinitial;
protected char getMiddleInitial()
{
   return middleInitial;
}
```

4. private access modifier

- Specifies that class members are only accessible by the class they are defined in.
- This means that no other class has access to private class members, even subclasses.

Classes and Objects

Objects

- An object is a self-contained element of a computer program that represents a related group of features and is designed to accomplish specific tasks.
- Also known as instances.
- Each object has a specific role in a program, and all objects can work with other objects in specifically defined ways.

Objects

More definitions for an Object

- A ‘thing’ may have a physical presence such as a ‘table’,‘chair’ or an abstract concept such as ‘a job’ .
- An object is an abstract representation of a ‘thing’ in the real world.
- We simulate a problem domain in the real - world through objects.
- An object has a unique identity, attributes (What it knows or data about it), and behavior (What it can do).

Objects

Employee Object Example

- For example, an Employee object (say employee1) will have the following attributes (what it knows):
  - name
  - age
  - salary
- It will also have the following behavior (what it can do):
  - set salary
  - get salary
  - set name
  - set age

Classes

- A Class is a template used to create multiple objects with similar features.
- When we write a program in an OO language, we don't need to define individual objects. Instead, we define classes of objects. Classes embody all features of a particular set of objects.
- For example, a Class "Tree" describes the features of all trees:
  - Has leaves & roots
  - Grows
  - Creates chlorophyll
- The Tree Class serves as an abstract model for the concept of a tree.
- Every class written in java made up of two components: Attributes (what it knows) and Behavior (what it can do)

Classes

1. Attributes

•Individual things that differentiate one class of objects from another and determine the appearance, state and other qualities of that class.

•– E.g: Color orange, raw umber, lemon yellow, maize

•Attributes of a class of objects also can include information about an object's state.

•In a class, attributes are defined by variables.

•Each object can have different values for its variables. These are called instance variables.

Classes

Instance variables and Class variables

- Instance variables
  - An instance variable is an item of information that defines an attribute of one particular object.
  - The object's class defines what kind of attribute it is, and each instance object stores its own value for that attribute.
  - Instance variables also known as object variables.
- Class variables
  - A Class variables is an item of information that defines an attribute of an entire class.
  - The variable applies to the class itself and to all of its instances.

Classes

2. Behavior

•Behavior is the way that a class of objects can do to change their attributes, and also what they do when ask them to do something.

•Examples for behavior

- Get angry
- Calm down
- Eat a peasant

•Behavior for a class of objects is done by using methods.

Classes

• Creating a Class

• Open the text editor to create Java programs. Start with class definition.

```
class Jabberwock
{
}
```

•Then create instance variables

```
String color;  boolean hungry;
```


Classes

• Creating a Class (continued)

• After that the programmer can add behavior to the class by adding methods.

```
void feedJabberwock()
{
if (hungry == true)
{
System.out.println("yum – a peasant");  hungry = false;
}
else
System.out.println("No, thanks – already ate");
}
```

•Use one of the following procedures to compile the program, depending on the system you're using.

```
javac Jabberwock.java
```


Classes

• Creating Objects

• Objects are created by instantiating classes. To use a class in a program, you must first create an instance of it. Objects of a class can be created using the new operator.

• Employee newEmp = new Employee();

• Object References following declaration will create an Object reference

• Employee newEmp2;

• You can create multiple References to the same object

• Employee newEmp; newEmp = new Employee(); newEmp2 = newEmp;

HNDIT Fundamentals of Programming

Week11: Object Oriented Programming concepts

Lesson introduction

- In this module, you will realize the importance of object-oriented thinking and learn how to design software using object-oriented techniques.

Introduction

- Structured and Object Oriented Anal are different techniques of developing a computer system.
- Structured
  - Is the traditional approach of software development based upon the waterfall model that focus is only on process and procedures.
- Object Oriented
  - Whereas in Object Oriented, the focus is more on capturing the real world objects in the current scenario that are of importance to the system.

What is object-orientation?

- In object-orientation, your program will be split into several small, manageable, reusable programs.
- Each small program has its own identity, data, logic and how it's going to communicate with the rest of the other small programs.

Object

- When you think about Objects, you should think about the real-world situations.
- Object Orientation was intended to be closer to the real world, hence, make it easier and more realistic.
- Objects can be modelled according to the needs of the application.
- An object may have a physical existence, like a car or an intangible conceptual existence, like a project.
- Each Objects has its own attributes, and behavior.
- Objects are separate from one another.
- They have their own existence, their own identity that is independent of other objects.

Object cont...


- Attributes
  - It's the characteristics or the properties of the Object
- Methods
  - Methods are the things that the Object can do




The concepts and terms in object orientation

Abstraction ,Encapsulation, Inheritance, Polymorphism

Encapsulation

- Encapsulation referrers to combining data and associated functions as a single unit.
- Binding (or wrapping) code and data together into a single unit is known as encapsulation.
- It also implies the idea of hiding the content of a Class, unless it is necessary to expose.
- We need to restrict the access to our class as much as we can so that we can change the properties and the behaviors only from inside the class

Encapsulation Example

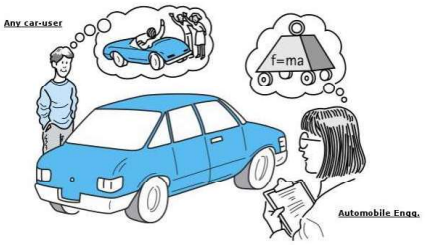
In java
status(variables)
and
behavior(methods)

related to a object are included into one single unit called as class

```
class student{
    String name;
    String indexNo;
    int age;

    public void setName(String name){
        this.name=name;
    }
    public String getName(){
        return name;
    }

    public void setIndexNo(String indexNo){
        this.indexNo=indexNo;
    }
    public String getIndexNo(){
        return indexNo;
    }
    public void setAge(int age){
        this.age=age;
    }
    public int getAge(){
        return age;
    }
}
```

Abstraction

- Hiding unnecessary details from the ordinary user.
- It handle complexity .
- That allows the user to perform more complex logic on top of the given abstraction without worrying about all the hidden complexity.



An abstraction includes the essential details relative to the perspective of the viewer

Example

- Making coffee with a coffee machine is an excellent example of abstractio
- A coffee machine user should only know how to use the coffee machine to make coffee, for instance, how to put water and coffee beans, switch it on and select the kind of coffee.
- The user does not need to know how the coffee machine is working internally on brewing a fresh cup of delicious coffee.
- Further, the user do not need to know the ideal temperature of the water or the amount of ground coffee.
- Someone else worried about that and created a coffee machine that now acts as an abstraction and hides all these details.
- You, as a coffee machine user, simply interact with a simple interface that doesn't require any knowledge about the internal implementation of the coffee machine.

Abstraction Example

In java interfaces contains method which does not have any method body

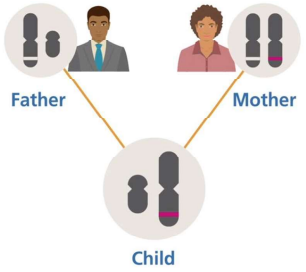
```
interface shape{

    final double pi=3.14;

    public void calArea();
    public void calCircumference();
}
```

Those methods should implement in it's sub classes.

```
class circle implements shape{

    double radius;

    public void calArea(){
       System.out.println("Area = " + (pi*radius*radius));
    }

    public void calCircumference(){
       System.out.println("Circumference = " + (2*pi*radius));
    }
}
```

Inheritance

- **Forming a new class from an existing class.**
- Existing class is base class/parent class/super class and new class is derived class/child class/sub class.
- Helps to reduce the code size



Example

- Different kinds of objects often have a certain amount in common with each other.
- For example:
  - Mountain bikes, road bikes, and tandem bikes, for example, all share the characteristics of bicycles (current speed, current pedal cadence, current gear).
  - Additional features that make them different: tandem bicycles have two seats and two sets of handlebars; road bikes have drop handlebars; some mountain bikes have an additional chaining, giving them a lower gear ratio.
  - Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.
  - In this example, Bicycle now becomes the superclass of Mountain Bike, Road Bike, and Tandem Bike

Inheritance

extends is the keyword used to inherit the properties of a class.

In the given program, when an object to **calculation** class is created, a copy of the contents of the superclass (**calculationSuper**) is made within it. That is why, using the object of the subclass you can access the members of a superclass.

The Superclass reference variable can hold the subclass object, but using that variable you can access only the members of the superclass, so to access the members of both classes it is recommended to always create reference variable to the subclass

```
class calculationSuper {
   int z;

   public void addition(int x, int y) {
      z = x + y;
      System.out.println("The sum of the given numbers:"+z);
   }

   public void Subtraction(int x, int y) {
      z = x - y;
      System.out.println("The difference between the given numbers:"+z);
   }
}

public class calculation extends calculationSuper {
   public void multiplication(int x, int y) {
      z = x * y;
      System.out.println("The product of the given numbers:"+z);
   }

   public static void main(String args[]) {
      int a = 20, b = 10;
      calculation demo = new calculation();
      demo.addition(a, b);
      demo.Subtraction(a, b);
      demo.multiplication(a, b);
   }
}
```

Polymorphism

- Polymorphism is the state where an object can take the shape of many different forms, and lets us do the right thing at the right time.
- It describes the concept that objects of different types can be accessed through the same interface
- Each type can provide its own, independent implementation of this interface.

Polymorphism

- *Overriding* and *overloading* are the core concepts in Java programming. They are the ways to implement polymorphism in our Java programs.

Overriding

When the method signature (name and parameters) are the same in the superclass and the child class, it's called *overriding*.

If a class inherits a method from its superclass, then there is a chance to override the method provided that it is not marked final.

The benefit of overriding is: ability to define a behavior that's specific to the subclass type, which means a subclass can implement a parent class method based on its requirement.

```
class Animal {
   public void move() {
      System.out.println("Animals can move");
   }
}

class Dog extends Animal {
   @Override
   public void move() {
      System.out.println("Dogs can walk and run");
   }
}

public class TestDog {

   public static void main(String args[]) {
      Animal a = new Animal();   // Animal reference and object
      Animal b = new Dog();   // Animal reference but Dog object

      a.move();   // runs the method in Animal class
      b.move();   // runs the method in Dog class
   }
}
```

Overloading

When two or more methods in the same class have the same name but different parameters, it's called overloading.

```
class Animal {
   public void move() {
      System.out.println("Animals can move");
   }
   public void move(String animal) {
      System.out.println(animal + " can move");
   }
}

public class TestDog {

   public static void main(String args[]) {
      Animal a = new Animal();   // Animal reference and object


      a.move();   // runs the method in Animal class
      a.move("Cat");
   }
}
```

Q & A

Programming in JAVA

Swing and Event Handling

Event

- Change in the state of an object is known as event
- Event driven programming – develop application respond to user generated events
  - Button clicks
  - Mouse move
  - Keystrokes
- Events is represented in Java by a class that extends the AWTEvent class

Java Event Model

- **Source**
  - Object which event occurs.
  - Source providing information of the occurred event
- **Listener**
  - It is also known as event handler.
  - Listener is object generating response to an event.

java.awt.event

- Package defines classes and interfaces used for event handling in the AWT and Swing.

java.awt.event

- ActionListener
  - Interface for receiving action events.
  - Object created with is registered with a component, using the component‘ saddActionListener() method
  - When the action event occurs, that object's actionPerformed() method is invoked

```
import java.awt.*;
import java.awt.event.*;
public class tstEvnt implements ActionListener
        {
        private Frame F;
        private TextField T;
        private Button B;
        public static void main(String[] args)
                {
                tstEvnt T= new tstEvnt();
                T.setGUI();
                }
        public void setGUI()
                {
                 F = new Frame("Window Title");
                T= new TextField(20);
                B = new Button("Click me");
                F.setSize(350,100);
                F.setVisible(true);
                F.setLayout(new FlowLayout());
                F.add(T);
                F.add(B);
                B.addActionListener(this);//component registration
                }
         public void actionPerformed(ActionEvent e)
                {
                 T.setText("Button Clicked " );
                }
        }
```


```
import java.awt.*;
import java.awt.event.*;
public class tstEvnt implements ActionListener
        {
        private Frame F;
        private TextField T;
        private Button B;
        public static void main(String[] args)
                {
                tstEvnt T= new tstEvnt();
                T.setGUI();
                }
        public void setGUI()
                {
                 F = new Frame("Window Title");
                T= new TextField(20);
                B = new Button("Click me");
                F.setSize(350,100);
                F.setVisible(true);
                F.setLayout(new FlowLayout());
                F.add(T);
                F.add(B);
                B.addActionListener(this);//component registration
                }
         public void actionPerformed(ActionEvent e)
                {
                 T.setText("Button Clicked " );
                }
        }
```


OR

class tstEvent

```
import java.awt.*;
import java.awt.event.*;
public class tstEvnt implements ActionListener
        {
        private Frame F;
        private TextField T;
        private Button B;
                public void setGUI()
                {
                 F = new Frame("Window Title");
                T= new TextField(20);
                B = new Button("Click me");
                F.setSize(350,100);
                F.setVisible(true);
                F.setLayout(new FlowLayout());
                F.add(T);
                F.add(B);

        B.addActionListener(this);//component registration
                }
         public void actionPerformed(ActionEvent e)
                {
                 T.setText("Button Clicked " );
                }
        }
```

Class Event

```
public class Event
{
  public static void main(String[] args)
                {
                tstEvnt T= new tstEvnt();
                T.setGUI();
                }
}
```