Java identifiers

- Identifiers are the name given to variables, classes, methods, etc.

```
class Test
{
public static void main(String[] args)
{
int variable=10;
}
}
```

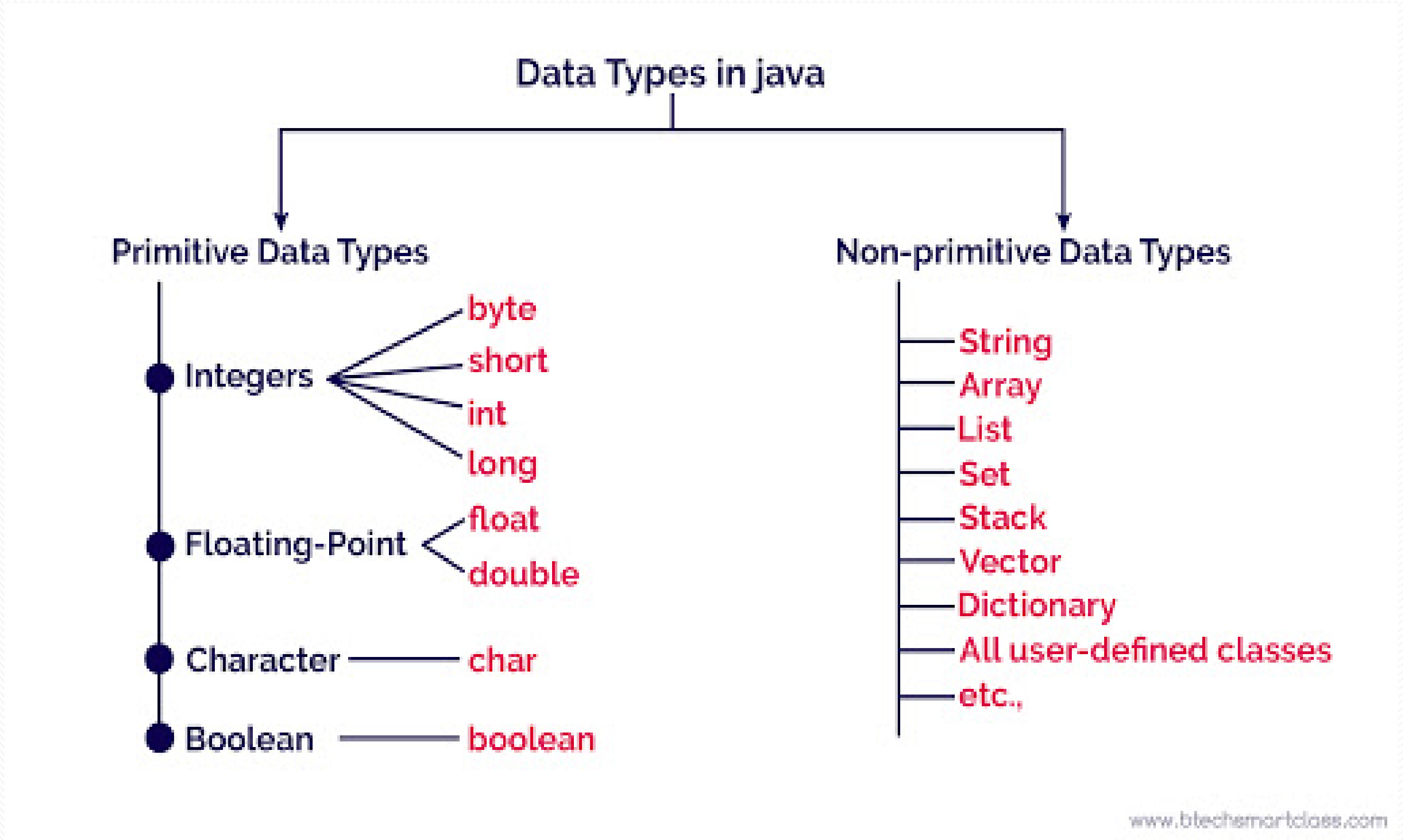
Rules for Naming an Identifier

- A to Z or a to z , long sequence of characters.
- Identifiers cannot be a keyword.
- Identifiers are case-sensitive.
- It can have a sequence of letters and digits. However, it must begin with a letter, $ or _. The first letter of an identifier cannot be a digit.

Keywords(50)

- **Access modifiers** –, public, private, protected. (3)
- **Class, method, variable modifiers**– abstract, class, extends, final, implements, interface, native, new, static, strictfp, synchronized, transient, volatile. (13)
- **Flow control**– break, case, continue, default, do, else, for, if, instanceof, return, switch, while. (12)
- **Package control**– import, package. (2)
- **Primitive types**– boolean, byte, char, double, float, int, long, short. (8)
- **Error handling**– assert, catch, finally, throw, throws, try. (6)
- **Enumeration**– enum. (1)
- **Others**– super, this, void. (3)
- **Unused**– const, goto. (2)

Data Types


Data Types in java
Primitive Data Types
Integers
byte
short
int
long
Floating-Point
float
double
Character
char
Boolean
boolean
Non-primitive Data Types
String
Array
List
Set
Stack
Vector
Dictionary
All user-defined classes
etc.,
www.btechsmartclass.com

Operators

Java divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators
- Conditional Operator

Arithmetic Operator

- Int x = 5
- Int y = 3
- System.out.print(x + y)   Addition 8
- print(x - y)   Subtraction   2
- print(x * y)   Multiplication 15
- print(x / y)   Division 1
- print(x % y)   Modulus   2

Assignment operators

x=5   Assignment   // 5

x+=3(x=x+3) **Add and Assign(+=) // 8**

X-=3(x=x-3) **Sub and Assign(-=) //   2**

X*=3(x=x*3) **Multiplication and Assign(*=) // 15**

X/=3(x=x/3) Division **and Assign(/=) / 1**

X%=5(x=x%3) **Modulus and Assign(%=) // 2**

Comparison operators

- int x=4, y = 3;

==	Equal	x == y	False
!=	Not equal	x != y	True
>	Greater than	x > y	True
<	Less than	x < y	False
>=	Greater than or equal to	x >= y	True
<=	Less than or equal to	x <= y	False

Logical operators

X=5

&& Returns True if both statements are true

- x < 15 && x < 10 True

|| Returns True if one of the statements is true

- x <= 5 || x > 14 False

Not

- Reverse the result, returns False if the result is true
- !(x <= 15 && x < 10) False

Bitwise Operator

- & AND
- Sets each bit to 1 if both bits are 1
- | OR
- Sets each bit to 1 if one of two bits is 1
- ^ XOR Sets each bit to 1 if only one of two bits is 1
- << leftshift
- >> rightshift

Increment Decrement

```
class PrePostDemo {
    public static void main(String[] args){
        int i = 3;
        i++;
        System.out.println(i);
        ++i;
        System.out.println(i);
        System.out.println(++i);
        System.out.println(i++);
        System.out.println(i);
    }
}
```

Types of variable

- Instance
- Static
- Local

Instance variable

```
class Student
{
String name;
int roll;
}
```

Instance variable

- If the value of a variable is varied from object to object such type of variable is called instance variables.
- For every object a separate copy of instance variables will be created.
- Instance variables should be declare with in the class directly.
-

Instance variable

- But outside of any method or block or constructor.
- Instance variable will be created at the time of object creation & destroyed at the time of object destruction.
- Instance variable will be stored in the heap memory as the part of object.

Static variable

- If the value of a variable is not varied from object to object such type of variable is called static variables.
- Static variables should be declare with in the class directly using static modifier. But outside of any method or block or constructor.
- In the case of static variable a single copy will be created at the class level and shared by every object of the class.

Static variable

- We can access static variable directly from both instance & the static area.

- For static variable JVM will always provide default value.

Local Variable

- Local variable will be declare inside of method or block or constructor.
- Such types of variables are called as local or temporary or stack or automatic.
- Local variables are stored in stack memory.

Local Variable

- Local variables will be created while execution the block, when the execution complete the variable is destroyed.

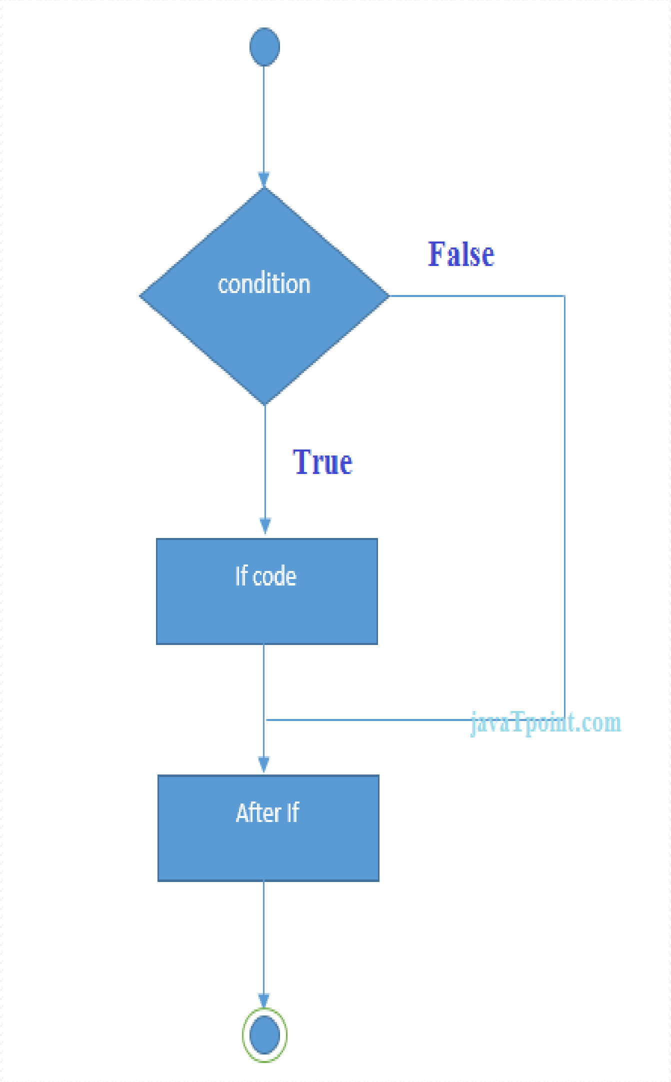
Control Statements

- if statement
- if-else statement
- if-else-if ladder
- nested if statement

Simple if

Syntax:

```
if(condition){
//code to be executed
}
```



Simple if

```
public class IfExample {
public static void main(String[] args) {

    int age=20;

    if(age>=18){
        System.out.print("Age is greater than 18");
    }
}
}
```

if-else

```
public class IfElseExample {
public static void main(String[] args) {

    int number=13;

    if(number%2==0){
        System.out.println("even number");
    }else{
        System.out.println("odd number");
    }
}
```

if-else-if ladder

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Nested if

Syntax:

```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed
   }
}
```

Nested if

```
public class JavaNestedIfExample {
public static void main(String[] args) {

    int age=20;
    int weight=80;

    if(age>=18){
        if(weight>50){
            System.out.println("You are eligible to donate blood");
        }
    }
}}
```

Loops

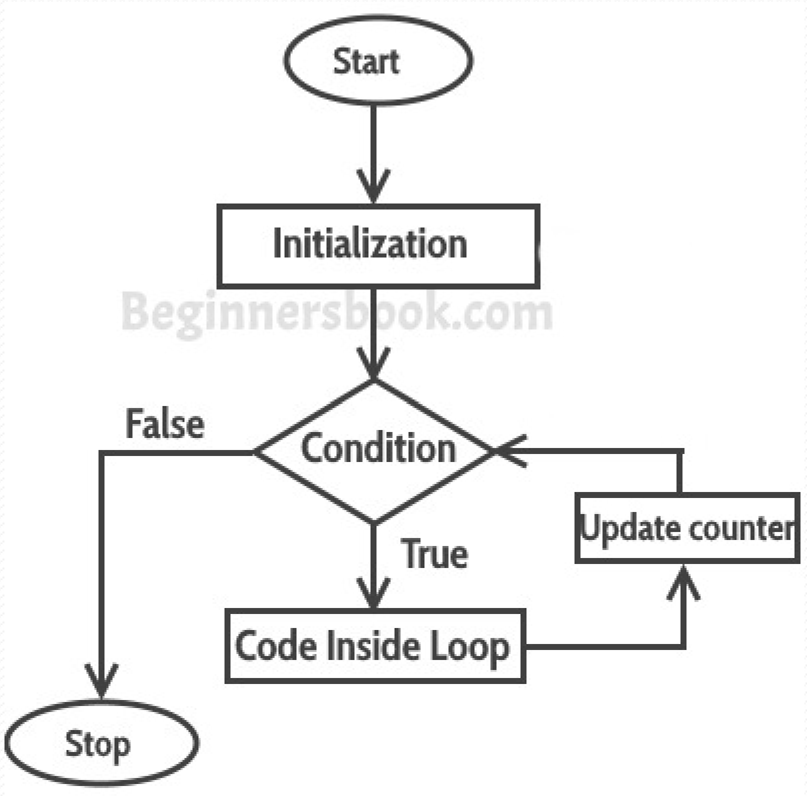
- Loops are used to execute a set of instructions/ functions repeatedly when some conditions become true.

- for loop
- while loop
- do-while loop

For loop

- The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

Simple for loop



Simple for loop

```
public class ForExample {
public static void main(String[] args) {

    for(int i=1;i<=10;i++)
{
        System.out.println(i);
    }
}
}
```

For loop example to iterate an array

```
class ForLoopExample3
 {
 public static void main(String args[])
{
 int arr[]={2,11,45,9};
for(int i=0; i<arr.length; i++)
{
 System.out.println(arr[i]);
 }
}
}
```

Break keyword

```
public class BreakExample {
public static void main(String[] args) {
    //using for loop
    for(int i=1;i<=10;i++){
        if(i==5){
            //breaking the loop
            break;
        }
        System.out.println(i);
    }
}
}
```

Continue keyword

```
public class ContinueExample {
public static void main(String[] args) {
    //for loop
    for(int i=1;i<=10;i++){
        if(i==5){
            //using continue statement
            continue;//it will skip the rest statement
        }
        System.out.println(i);
    }
}
}
```

While Loop

- **Syntax of the *While* Loop**
- The syntax of a *while* loop is as follows:
- while(BooleanExpressionHere)
- {
- YourStatementHere
- }

Example – 1

```
public static void main(String args[])
    {
            int num = 0;
            System.out.println("Let's count to 10!");
            while(num < 10)
            {
                  num = num + 1;
                  System.out.println("Number: " + num);
            }
System.out.println("We have counted to 10! Hurray! ");
      }
```

do-while Loop

Syntax:

```
do{
//code to be executed
}while(condition);
```

Example – 1

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

Methods

As we discussed above, an object has attributes and behaviours. These behaviours are called methods in programming.

No Argument No Return

```
public class NewClass6 {
    void sum()
    {
        System.out.println("add"+(4+2));
    }
    public static void main(String[] args) {

NewClass6 nsix=new NewClass6();
nsix.sum();
     }}
```

With Argument No Return

```
import java.util.Scanner;
public class NewClass6 {
    void sum(int a, int b)
    {
        System.out.println("add"+(a+b));
    }
    public static void main(String[] args) {
    Scanner s= new Scanner(System.in);
    NewClass6 nsix=new NewClass6();
    int a,b;
    System.out.println("Enter the values");
    a=s.nextInt();
    b=s.nextInt();
    nsix.sum(a,b);
     }
}
```

With Argument With Return

```
 import java.util.Scanner;
public class NewClass6 {
    int sum(int a, int b)
    {
        return a+b;
    }
    public static void main(String[] args) {
    Scanner s= new Scanner(System.in);
    NewClass6 nsix=new NewClass6();
    int a,b;
    System.out.println("Enter the values");
    a=s.nextInt();
    b=s.nextInt();
   System.out.println( nsix.sum(a,b));
     }
}
```

No Argument With Return

```
public class NewClass6 {
    int sum( )
    {
        int a=33;
        return  a;
    }
    public static void main(String[] args) {

    NewClass6 nsix=new NewClass6();

   System.out.println( nsix.sum( ));
     }
}
```

OOPs Concepts

- The object is related to real-word entities such as book, house, pencil, etc.
- we can easily create and use classes and objects
- The oops concept focuses on writing the reusable code.

Concepts in OOPS

- Class
- Object
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

- A class is a blueprint for the objects.

For example: Ram, Shyam, Steve, Rick are all Objects, so we can define a template (blueprint) class Human for these objects.

The class can define the common attributes and behaviors of all the objects.

Class

- A class is a blueprint for the objects.
- **For example**: Ram, Shyam, Steve, Rick are all objects so we can define a template (blueprint) class Human for these objects.
- The class can define the common attributes and behaviours of all the objects.

Constructors

- A constructor is a block of codes similar to the method.
- It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
- It's name is same as class name and it does not return any value.
- It calls a default constructor if there is no constructor available in the class.
-

There are two types of constructors

- no-arg constructor, and parameterized constructor.

Default Constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.

```
public class NewClass6 {
   NewClass6()
         {
            System.out.println("i'm the default constructor");
         }
   public static void main(String[] args) {

   NewClass6 nsix=new NewClass6();


    }
}
```

Default Constructor

```
public class NewClass6 {
    int id;
String name;

void display()
{
    System.out.println(id+" "+name);
}

public static void main(String args[]){

NewClass6 s1=new NewClass6();
NewClass6 s2=new NewClass6();

s1.display();
s2.display();
}

}
```

Parameterized Constructor

- A constructor which has a specific number of parameters is called a parameterized constructor.

Parameterized Constructor

```
public class NewClass6 {
      int id;
    String name;

    NewClass6(int i,String n){
    id = i;
    name = n;
    }

    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){

    NewClass6 s1 = new NewClass6(111,"Sai");
    NewClass6 s2 = new NewClass6(222,"Swarna");

    s1.display();
    s2.display();
    }
}
```

Constructor Overloading

- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

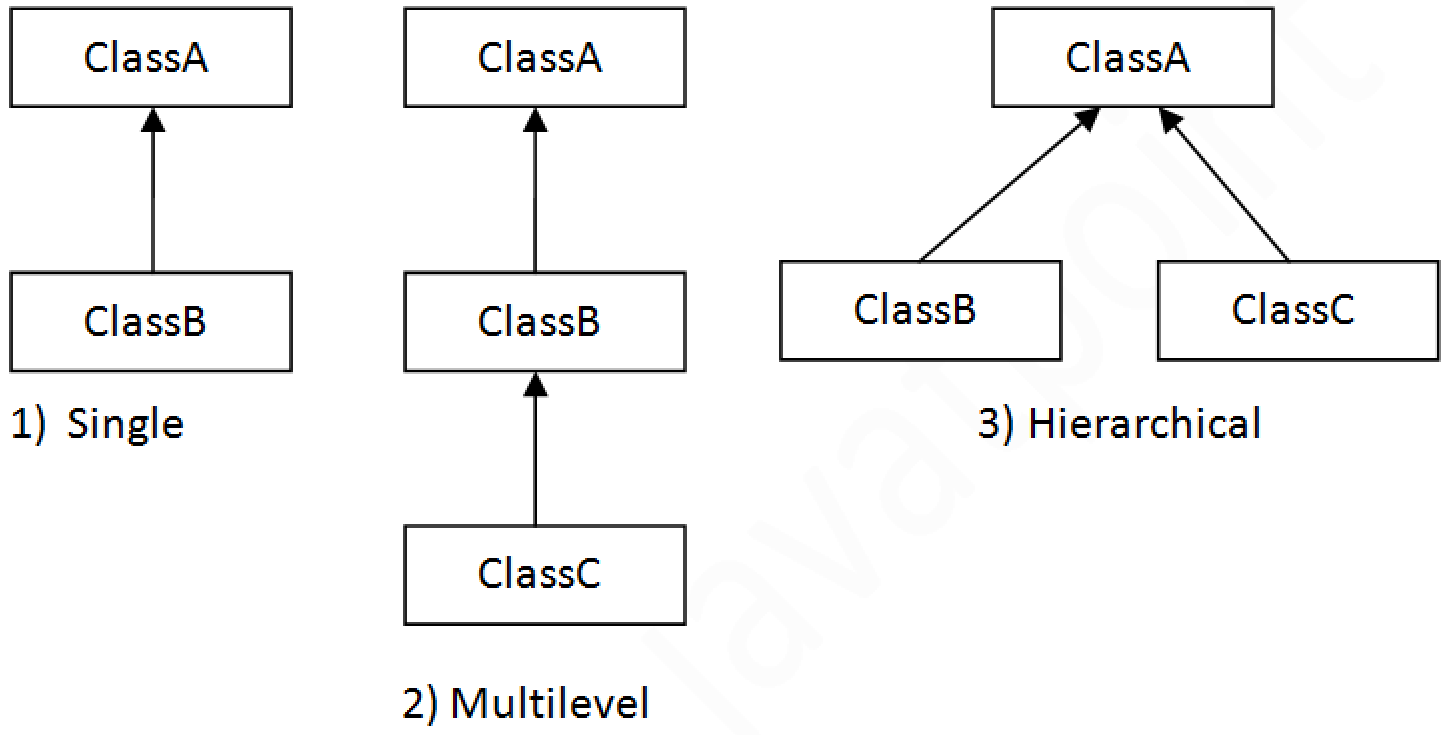
Constructor Overloading

```
public class NewClass6
{
      int id;
    String name;
    int age;

    NewClass6(int i,String n){
    id = i;
    name = n;
    }

    NewClass6(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
    void display(){System.out.println(id+" "+name+" "+age);}

    public static void main(String args[]){
    NewClass6 s1 = new NewClass6(111,"Sai");
    NewClass6 s2 = new NewClass6(222,"Swaena",25);
    s1.display();
    s2.display();
   }
}
```

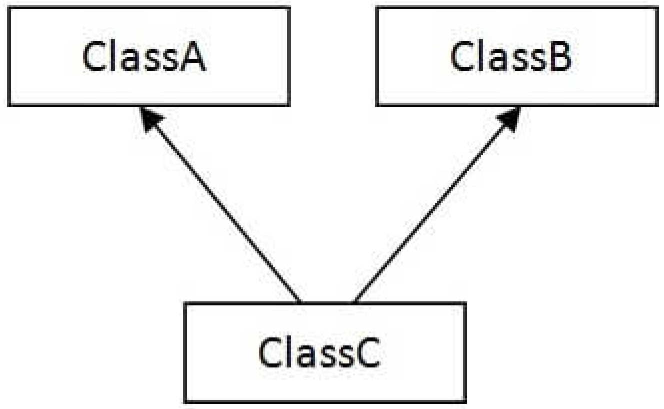
Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- **Parent class** is the class being inherited from, also called base class.
- **Child class** is the class that inherits from another class, also called derived class

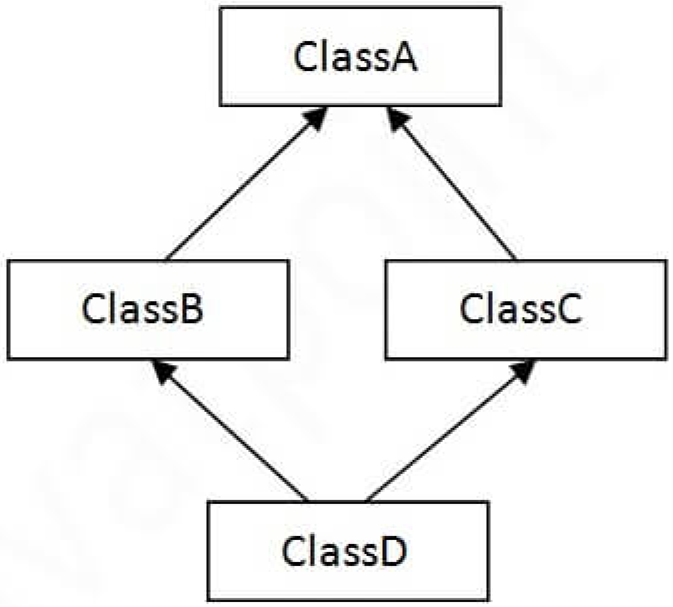
Types of inheritance



Types of inheritance



4) Multiple



5) Hybrid

Single

```
public class Parent {
    void car()
    {
        System.out.println("Parent having car");
    }
    void house()
    {
        System.out.println("Parent having house");
    }
}
class child extends Parent
{
     void bike()
    {
        System.out.println("child having bike");
    }
    public static void main(String[] args) {
        child c = new child();
        c.bike();
        c.car();
        c.house();
    }

}
```

Multi Level

```
import java.util.Scanner;
class Employee{
  int empId;
    String name;
    public void setData(int c,String d){
        empId=c;
        name=d;
    }
    public void showData(){
        System.out.print("EmpId = "+empId + "  " + " Employee Name = "+name);
        System.out.println();
    }
}
public class NewClass6 {
       public static void main(String args[]){
     int id;
     String name;
     Scanner s = new Scanner(System.in);

   int t;
           System.out.println("Enter the no of records");
           t=s.nextInt();
     Employee[] obj = new Employee[t] ;

     for (int i=0; i<obj.length;i++)
     {
      obj[i] = new Employee();
      System.out.println("Enter the id");
      id= s.nextInt();
      System.out.println("Enter the name");
     name=s.next();
     obj[i].setData(id,name);

     }
          for (int i=0; i<obj.length;i++)
     {
     System.out.println("Employee Object :"+i);
     obj[i].showData();

     }
  }
}
```

Hierarchical

```
public class Parent   {
   void car()
   {
      System.out.println("Parent having car");
   }
   void house()
   {
      System.out.println("Parent having house");
   }
}
class child1 extends Parent
{
    void bike()
   {
      System.out.println("child having bike");
   }
   public static void main(String[] args) {
      child1 c = new child1();
      c.bike();
      c.car();
      c.house();
      child2 c2 = new child2();
      c2.cycle();
      c2.car();
      c2.house();

   }}
   class child2 extends Parent
{

    void cycle()
   {
      System.out.println("child having cycle");
   }



   }
```

Java Interface

```
interface FirstInterface {
  public void myMethod(); // interface method
}
interface SecondInterface {
  public void myOtherMethod(); // interface method
}
class DemoClass implements FirstInterface, SecondInterface {
  public void myMethod() {
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}
public class ClasMain {
    public static void main(String[] args) {
    DemoClass myObj = new DemoClass();
    myObj.myMethod();
    myObj.myOtherMethod();
    }
}
```

Polymorphism

- **Polymorphism in Java** is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
- There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

Runtime Polymorphism

```
class Bike{
  void run(){System.out.println("running");}
}
class Splendor extends Bike{
  void run(){System.out.println("running safely with 60km");
   }

  public static void main(String args[]){
    Bike b = new Splendor();//upcasting
    b.run();
  }
}
```

Runtime Polymorphism

```
class Bank{
float getRateOfInterest(){return 0;}
}
class SBI extends Bank{
float getRateOfInterest(){return 8.4f;}
}
class ICICI extends Bank{
float getRateOfInterest(){return 7.3f;}
}
class AXIS extends Bank{
float getRateOfInterest(){return 9.7f;}
}
class TestPolymorphism{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
b=new ICICI();
System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
b=new AXIS();
System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());
}
}
```

Compile time polymorphism

- **Method Overloading :**
  If a class have multiple methods by same name but different parameters, it is known as Method Overloading.

Method Overloading

```
class Calculation{
  void sum(int a,int b)
  {
     System.out.println(a+b);
  }
  void sum(int a,int b,int c){
     System.out.println(a+b+c);
   }

  public static void main(String args[]){
  Calculation obj=new Calculation();
  obj.sum(10,10,10);
  obj.sum(20,20);

  }
}
```

Abstraction

- A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Abstraction

```
abstract class Bike{
  abstract void run();
}

public class Honda extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda();
 obj.run();
}

}
```

Abstraction

```
abstract class Shape1{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle1 extends Shape1{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape1{
void draw(){System.out.println("drawing circle");}
}

public class TestAbstract {
    public static void main(String args[]){
Shape1 s=new Circle1();//In a real scenario, object is provided through method, e.g., getShape()
      method
s.draw();
}

}
```

Encapsulation

- **Encapsulation in Java** is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.
- We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

Encapsulation

```
class Student{

private String name;

public String getName(){
return name;
}
//setter method for name
public void setName(String name){
this.name=name ;
}
}
public class Encaps {
    public static void main(String[] args){
Student s=new Student();
 s.setName("Sainath");
   System.out.println(s.getName());
}
}
```

package

- **What is Package in Java?**
- **PACKAGE in Java** is a collection of classes, sub-packages, and interfaces. It helps organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve code reusability.
- Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.

Types Of Package

- Package can be built-in and user-defined, Java provides rich set of built-in packages in form of API that stores related classes and sub-packages.
- **Built-in Package:** math, util, lang, i/o etc are the example of built-in packages.
- **User-defined-package:** Java package created by user to categorize their project's classes and interface are known as user-defined packages.

Exception Handling

- The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.
-

Types of Java Exceptions

- There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:
- Checked Exception
- Unchecked Exception
- Error

Difference between Checked and Unchecked Exceptions

- 1) Checked Exception
- The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.
- 2) Unchecked Exception
- The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.
- 3) Error
- Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Example

```
public class TryCatchExample1 {
    public static void main(String[] args) {
      int data=50/0; //may throw exception
      System.out.println("rest of the code");
  }  }
```

```
public class TryCatchExample2 {

    public static void main(String[] args) {
        try
        {
        int data=50/0; //may throw exception
        }
            //handling the exception
        catch(ArithmeticException e)
        {
            System.out.println(e);
        }
        System.out.println("rest of the code");
    }

}
```

Multi-catch block

```
public class MultipleCatchBlock1 {

    public static void main(String[] args) {

          try{
               int a[]=new int[5];
               a[5]=30/0;
              }
              catch(ArithmeticException e)
                {
                 System.out.println("Arithmetic Exception occurs");
                }
              catch(ArrayIndexOutOfBoundsException e)
                {
                 System.out.println("ArrayIndexOutOfBounds Exception occurs");
                }
              catch(Exception e)
                {
                 System.out.println("Parent Exception occurs");
                }
              System.out.println("rest of the code");
    }
}
```

Nested Block

```
class Excep6{
 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);}

    try{
    int a[]=new int[5];
    a[5]=4;
    }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

  }catch(Exception e){System.out.println("handeled");}

  System.out.println("normal flow..");
 }
}
```

Finally block

- **Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc.
- Java finally block is always executed whether exception is handled or not.
- Java finally block follows try or catch block.

Finally

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.
   println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Finally

```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.
   println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Finally

```
public class TestFinallyBlock2{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(ArithmeticException e){System.out.println(e);}
  finally{System.out.
   println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Throw keyword

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

Throw

```
public class TestThrow1{
   static void validate(int age){
     if(age<18)
      throw new ArithmeticException("not valid");
     else
      System.out.println("welcome to vote");
   }
   public static void main(String args[]){
      validate(13);
      System.out.println("rest of the code...");
  }
}
```

Throws keyword

```
import java.io.IOException;
class Testthrows1{
  void m()throws IOException{
    throw new IOException("device error");//checked exception
  }
  void n()throws IOException("device errorw");{
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handled");}
  }
  public static void main(String args[]){
   Testthrows1 obj=new Testthrows1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

Applet

- An applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server.

Lifecycle of Java Applet

- Applet is initialized.
- Applet is started
- Applet is painted.
- Applet is stopped.
- Applet is destroyed.

Example – 1

```
package javaappletsprg;
import java.awt.*;
import java.applet.*;
public class JavaApplets extends Applet {
    public void paint(Graphics g)
    {
      g.drawString("A simple Applet", 20, 20);
    }

}
```

Applet

- Every Applet application must import two packages - java.awt and java.applet.
- java.awt.* imports the Abstract Window Toolkit (AWT) classes.
- Applets interact with the user (either directly or indirectly) through the AWT.
- The AWT contains support for a window-based, graphical user interface.

- java.applet.* imports the applet package, which contains the class Applet.
- Every applet that you create must be a subclass of Applet class.
-

How to create the applet class file

- The class in the program must be declared as public, because it will be accessed by code that is outside the program.
- Every Applet application must declare a paint() method.
- This method is defined by AWT class and must be overridden by the applet.
- The paint() method is called each time when an applet needs to redisplay its output.
- Execution of an applet does not begin at main() method. In fact an applet application does not have any main() method.