5. OOPs

Data Hiding:

Our Internal Data should Not go out Directly OR Outside Person can't Access Our Internal Data Directly. This is the Concept of Data Hiding.

Eg: After Providing Proper User Name and Password Only we can able to Access Our Mail Information.

Eg: After swiping ATM Card and Providing Valid Pin Number we can able to Access Our Account Information.

By declaring Data Member as private we can achieve Data Hiding.

```
class Account {
      private double balance;
       ::::::::::::::::::::::::
       ::::::::::::::::::::::::
}
```

The Main Advantage of Data Hiding is Security.

Note: Recommended Modifier for Data Members is private.

Abstraction:

Hiding Internal Implementation and Highlight the Set of Services which are offering is the Concept of Abstraction.

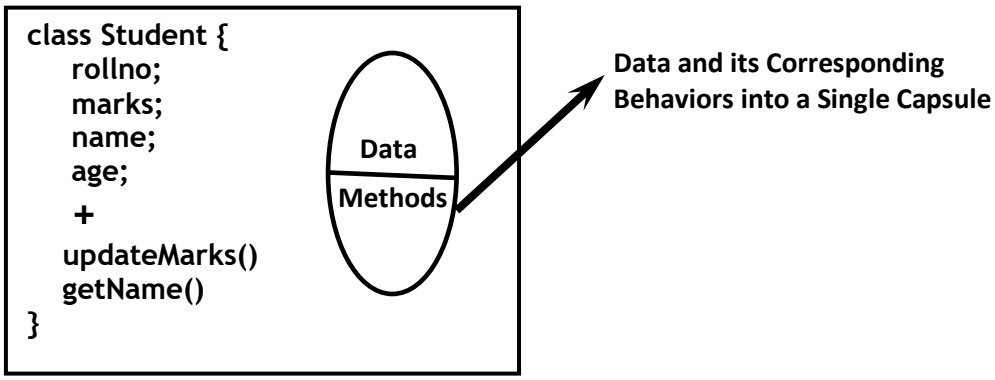
Eg: By using Bank ATM GUI Screen Bank People are highlighting the Set of Services what they offering without highlighting Internal Implementation.

The Main Advantages of Abstraction are

- We can Achieve Security as we are not highlighting Our Internal Implementation.
- Without effecting Outside Person we can able to Perform any Type of Changes in Our Internal Design. Hence Enhancement will become Easy.
- It Improves Maintainability and Modularity of the Application.

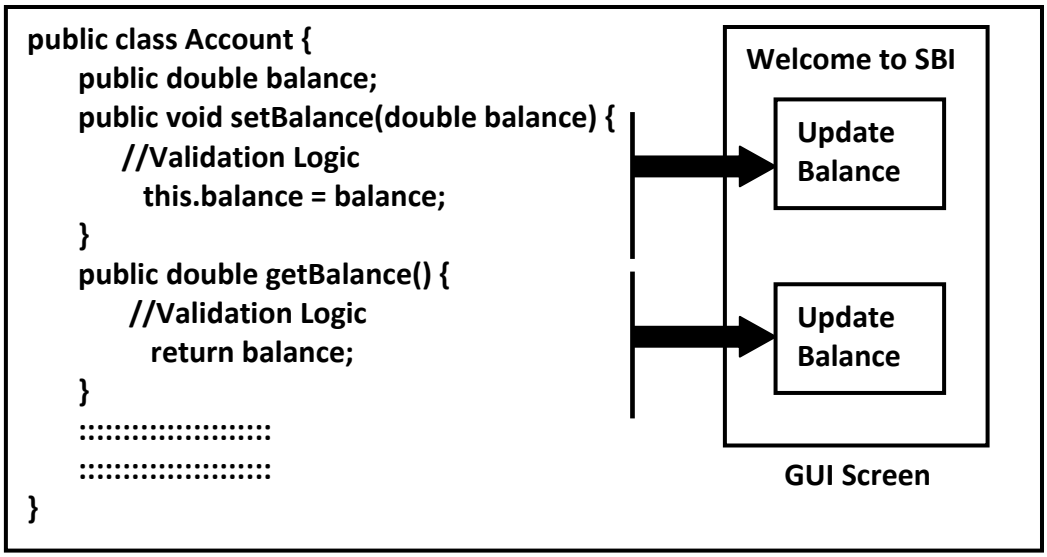
Encapsulation:

The Process of Binding Data and Corresponding Methods into a Single Unit is Called Encapsulation.



In any Component follows Data Hiding and Abstraction Such Type of Component is Called Encapsulated Component.

Encapsulation = Data Hiding + Abstraction



The Main Advantages of Encapsulation are

- We can Achieve Security.
- Enhancement will become Very Easy.
- It Improves Maintainability and Modularity of the Application.

The Main Advantage of Encapsulation is we can Achieve Security and the Main Disadvantage of Encapsulation is it Increases Length of the Code and Slow Down Execution. So that Performance will be Impacted.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Tightly Encapsulated Class:

- A Class is Said to be Tightly Encapsulated if and only if Each and Every Variable of that Class declared as private.
- Whether the Class contains Getter and Setter Methods OR Not and whether these Methods are declared as public OR Not. These things are not required to Check.

```
public class Account {
        private double balance;
        public double getBalance() {
                return balance;
        }
}
```

Which of the following Classes are Tightly Encapsulated?

```
class A {
   private int x = 10;
}

class B extends A {
    int y = 10;
}

class C extends A {
    private int z = 20;
}
```

Class A and Class C are Said to be Tightly Encapsulated Classes

Which of the following Classes are Tightly Encapsulated?

```
class A {
     int x = 10;
}

class B extends A {
    private int y = 20;
}

class C extends B {
    private int z = 30;
}
```

No Class is Tightly Encapsulated Classes.

Note: If the Parent Class is Not Tightly Encapsulated then No Child Class is Tightly Encapsulated.

IS-A Relationship (Inheritance):

- By using extends Key Word we can implement IS-A Relationship.
- The Main Advantage of Inheritance is Re-usability of the Code.

```
class P {
    public void m1() {}
}

class C extends P {
    public void m2() {}
}

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

        //Case 1
        P  p = new P();
        p.m1();
        p.m2();  cannot find symbol
                 symbol:   method m2()
                 location: variable p of type P

        //Case 2
        C c = new C();
        c.m1();
        c.m2();

        //Case 3
        P  p = new C();
        p.m1();
        p.m2();  cannot find symbol
                 symbol:   method m2()
                 location: variable p of type P

        //Case 4
        C c = new P();  error: incompatible types
                               required: C
                               found:    P
    }
}
```

Conclusions:

1) Whatever Methods Parent has by Default Available to the Child. Hence on Child Class Object we can Call Both Parent and Child Class Methods.

2) Whatever Methods Child has by Default Not Available to the Parent and Hence on the Parent Class Reference we can't Call Child Specific Methods.

3) Parent Reference can be used to hold Child Object. But by using that Reference we can Call Only Methods Available in Parent Class and we can't Call Child Specific Methods.

4) Child Reference cannot be used to hold Parent Object. But Parent Reference can be used to hold Child Object.

5) Parent Class contains the Common Functionality which required for Child Class. Whereas Child Class contains Specific Functionality.

Without Inheritance

```
class HousingLoan {
    300 Methods
}
```

```
class PersonalLoan {
    300 Methods
}
```

```
class EducationalLoan {
    300 Methods
}
```

900 Methods

90 Hours for Development

With Inheritance

```
class Loan {
    250 Methods
}
```

```
class HousingLoan  extends Loan {
    50 Methods
}
```

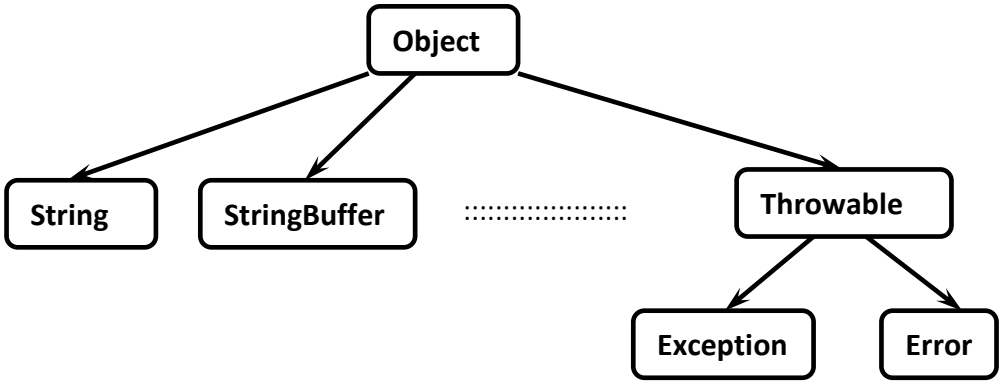
```
class PersonalLoan  extends Loan {
    50 Methods
}
```

```
class EducationalLoan  extends Loan {
    50 Methods
}
```

400 Methods
40 Hours for Development

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Note: Total Java API is implemented by using Inheritance Concept Only.



- The Most Common Methods which are required for All Java Classes are defined Inside Object Class. Hence Object Class Acts as Root for All Java Classes.
- The Most Common Methods which are required for All Exceptions and Errors are defined in Throwable Class. Hence Throwable Class Acts as Root for Exception Hierarchy.

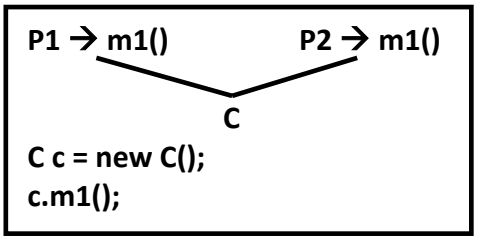
Multiple Inheritance:

A Java Class can't extend More than One Class at-a-Time. Hence Java won't Provide Support for Multiple Inheritance with Respect to Classes.

Eg: class A extends B, C { } ✗

Why Java won't Provide Support for Multiple Inheritance?

- There May be a Chance of Raising Ambiguity Problems.

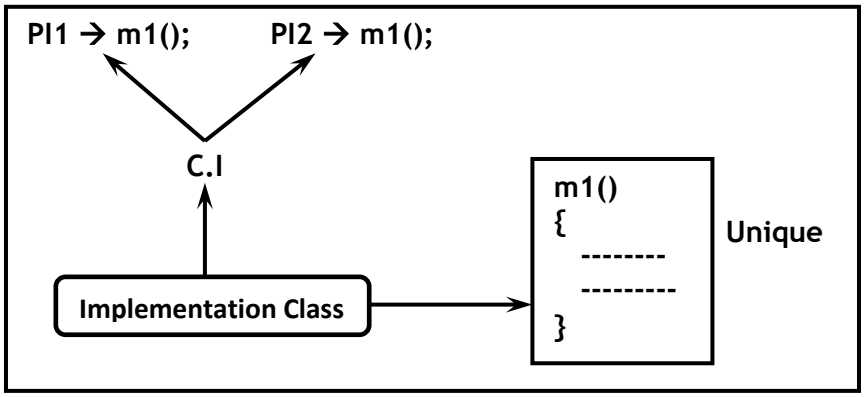


- An Interface can extend any Number of Interfaces at a Time. Hence Java Provides Support for Multiple Inheritance with Respect to Interfaces.

```
interface A {}      interface B {}
interface C extends A, B {} √
```

Why Ambiguity Problem won't be raised in Interfaces?

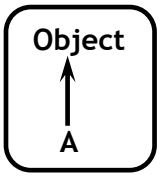
Even though Multiple Method Declarations Present, but Implementation is Unique. Hence there is No Chance of Ambiguity Problem in Interfaces.



Note:

If Our Class doesn't extends Any Other Class then Only it is the Direct Child Class of Object.

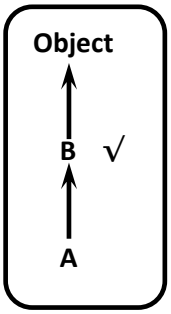
Eg: class A {}



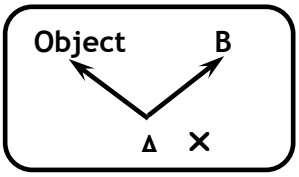
If Our Class extends Any Other Class then it is the In- Direct Child Class of Object.

Eg: class A extends B {}

Multi- Level Inheritance



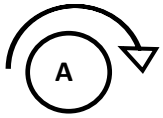
Multiple Inheritance



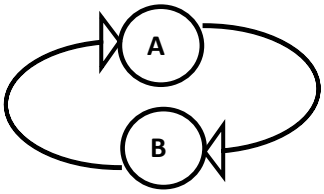
Cyclic inheritance:

Cyclic Inheritance is Not allowed in Java.

Eg: class A extends A {}
CE: cyclic inheritance involving A



Eg: class A extends B {}
class B extends A {}
CE: cyclic inheritance involving A



DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

HAS-A Relationship:

- It is Also Known as *Composition* OR *Aggregation*.
- There is No Specific Key Word to implement HAS-A Relationship. But Most of the Times we are depending on *new* Key Word.
- The Main Advantage of HAS-A Relationship is Code Re- Usability.
- The Main Disadvantage is depending between the Components and Creates Maintenance Problems.

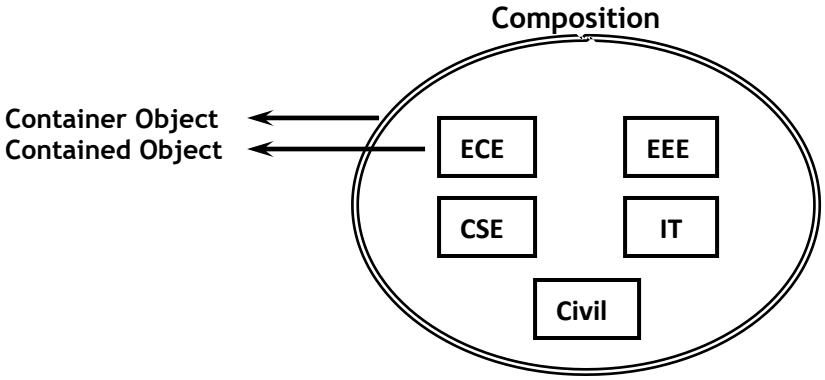
```
class Car {
      Engine e = new Engine();
                  .
                  .
} //Class Car has engine Reference

class Engine {
    //Engine Specific
Functionality.
}
```

Composition Vs Aggregation

Composition

Without Existence of Container Object, if there is No Chance of Existence of Contained Objects then *Container* and *Contained* Objects are Said to be Strongly Associated and this *Strong Association* is known as Composition.

Eg: A University has Several Departments. Without Existence of University there is No Chance for Existence Departments Objects. Hence *University* and *Departments* are Strongly Associated and this Strong Association is Known as *Composition*.

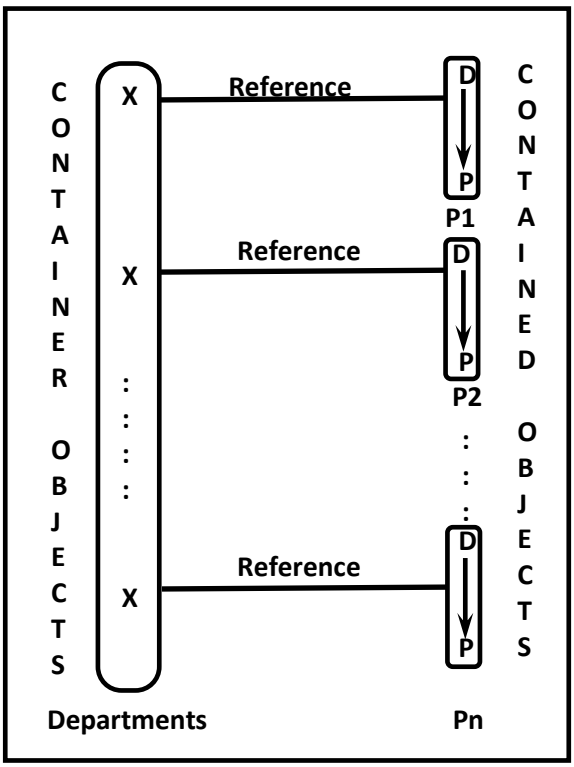


Aggregation

Without Existence of Container Object, if there is a Chance of Existence of Contained Objects then *Container* and *Contained* Objects are Said to be Weakly Associated and this *Loose Association* is known as Aggregation.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Eg: Within a Department there May be a Chance of working Several Professors. Without Existence of Departments Object there May be a Chance of existing Professors Object. Hence Department and Professors are Loosely Associated and this *Loose Association* is Known as Aggregation.

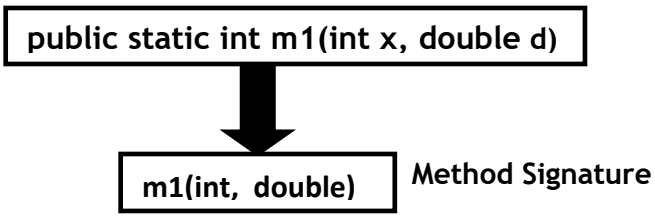


Note:

- In Composition Objects are Strongly Associated whereas in Aggregation Objects are Weakly Associated.
- In Composition Container Object holds Contained Objects Directly whereas in Aggregation Container Object Just Holds References of Contained Objects.

Method Signature:

- Method Signature consists of *Method Name* and *Argument Types*.



- Return Type is Not Part of Method Signature in Java.
- Compiler will Use Method Signature while resolving Method Calls.


```
class Test {
     public void m1(int i) {------}
     public void m1(String s) {------}
}
Test t1 = new Test();
    t1.m1(10);
    t1.m1("Durga");
    t1.m1(10.9); //CE: cannot find symbol
                        symbol: m1(double)
                        location: class Test
```

Test

m1(int)

m1(String)

Method Signature

- Within a Class 2 Methods with Same Signature Not allowed. Otherwise we will get Compile Time Error Saying m1() is already defined in Test class.

```
class Test {
        public void m1(int i) {}
        public int m1(int x) {
             return 10;
        }
}

     Test t1 = new Test();
      t1.m1(10);
CE: m1(int) is already defined in Test
```

m1(int)

m1(int)

Overloading

- 2 Methods are Said to be Overloaded if and only if Both Methods having Same Name but Different Type of Arguments.
- In C Language Overloading Concept is Not there hence we can't Declare 2 Methods with the Same Name but different Type of Arguments. Hence If there is a Change in Argument Type Compulsory we should go for New Method Name.

Eg: abs(int) →abs (10);
labs(long) → labs (10l);
fabs(float) → fabs (10.5f);

- Lack of Overloading in C Increases the Complexity of the Programming.
- But in Java we can Declare Multiple Methods with the Same Name but with different Arguments Types and these Methods are Called Overloaded Methods.

Eg: abs(int),
abs(long),
abs(float)

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Having Overloading Concept in Java Reduces the Complexity of the Programming.

Case 1:

The Overloading Method Resolution is the Responsibility of Compiler based on Reference Type and Method Arguments. Hence Overloading is Considered as *Compile-Time Polymorphism* OR *Early Binding*.

```
class Test {
    public void m1()
    {
        System.out.println ("no- args");
    }
    public void m1(int i)
    {
        System.out.println ("int- args");
    }
    public void m1(double d)
    {
        System.out.println ("double-args");
    }
    public static void main(String[] args) {
            Test t = new Test();
            t.m1();  → no-args
            t.m1(10); → int-args
            t.m1(10.5); → double-args
     }
}
```

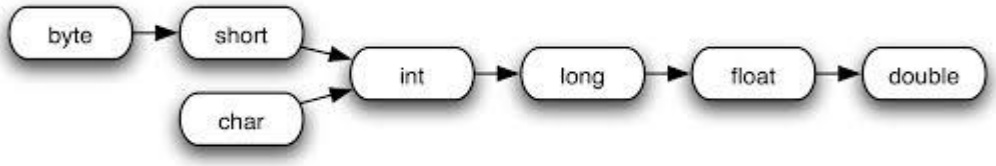
Overloaded Methods

Case 2: Automatic Promotion in Overloading

- While Resolving Overloaded Methods if Exact Method with the required Argument is Not Available then the Compiler won't Raise Immediately Compile Time Error.
- First Compiler will Promote Arguments to Next Level and Check is there any Matched Method with Promoted Arguments.
- If the Matched Method is Found then it will Considered Otherwise Compiler Promotes the Argument to the Next Level.
- This Process will be continued until all Possible Promotions.
- After all Possible Promotions still the Compiler Unable to find the Matched Method then it raises Compile Time Error.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

The following is the List of all Possible Automatic Promotions in Overloading.

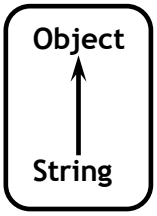



```
class Test {
    public void m1() {
        System.out.println ("no-args");
    }

    public void m1(int i) {
        System.out.println ("int-args");
    }

    public void m1(float f) {
        System.out.println ("float-args");
    }

    public static void main(String[] args) {
            Test t = new Test();

            t.m1();  //no-args

            t.m1(10);  //int-args

            t.m1(10.9f);  //float-args

            t.m1('a');  //int-args

            t.m1(10l);  //float-args

            t.m1(10.5);  CE: cannot resolve symbol
                                symbol  : method m1 (double)
                                location: class Test
    }
}
```

Case 3:

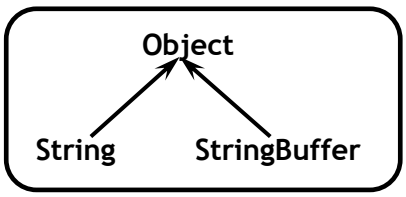
- In Overloading exact Match will get High Priority.
- In Overloading Child Class Argument will get More Priority than Parent Class Argument.

```
class Test {
    public void m1(String s) {
        System.out.println ("String Version");
    }
    public void m1(Object o) {
        System.out.println ("Object Version");
    }
    public static void main(String arg[]) {
            Test t = new Test();
            t.m1("Durga"); //String Version
            t.m1(new Object()); //Object Version
            t.m1(null); //String Version
      }
}
```



Case 4:
In Java, Method Overloading is Not Possible by Changing the Return Type of the Method because there May Occur Ambiguity Problem.

```
class Test {
   public void m1(String s) {
         System.out.println ("String Version");
   }

   public void m1(StringBuffer sb) {
         System.out.println ("StringBuffer Version");
   }

   public static void main(String arg[]) {
         Test t = new Test();
         t.m1("Durga"); //String Version
         t.m1(new StringBuffer("Durga")); //StringBuffer Version
         t.m1(null); //CE: reference to m1 is ambiguous, both method m1(String) in Test and
                              method m1(StringBuffer) in Test match
      }
}
```



Case 5: In General var-arg Method will get Least Priority i.e., if No Other Method Matched then Only var-arg Method will get the Chance. It is Exactly Same as *default* Case Inside *switch*.

```
class Test {
   public void m1(int i) {
         System.out.println ("General Method");
   }
   public void m1(int... i) {
         System.out.println ("var-arg Method");
   }
   public static void main(String arg[]) {
               Test t = new Test();
               t.m1(); //var-arg Method
               t.m1(10, 20); //var-arg Method
               t.m1(10); //General Method
    }
}
```

Case 6:

```
class Test {
    public void m1(int i, float f) {
          System.out.println("int - float Version");
    }
    public void m1(float f, int i) {
         System.out.println("float - int Version");
    }
    public static void main(String arg[]) {
          Test t = new Test();
           t.m1(10.5f, 10); //float - int Version
           t.m1(10, 10.5f); //int - float Version
           t.m1(10, 10); // C.E: reference to m1 is ambiguous, both method m1(int,float) in Test and
                                method m1(float,int) in Test match
           t.m1(10.5f, 10.5f); // C.E: cannot resolve symbol
                                       symbol  : method m1 (float,float)
                                       location: class Test
     }
}
```

Case 7:

Overloading Method Resolution will always take Care by Compiler based on the Reference Type but not based on Runtime Object.

```
class Animal {}
class Monkey extends Animal {}
class Test {
    public void m1(Animal a) {
        System.out.println ("Animal Version");
    }
    public void m1(Monkey m) {
        System.out.println ("Monkey Version");
    }
    public static void main(String arg[]) {
        Test t = new Test();
        Animal a = new Animal();
        t.m1(a); //Animal Version

        Monkey m = new Monkey();
        t.m1(m); //Monkey Version

        Animal a1 = new Monkey();
        t.m1(a1); //Animal Version
    }
}
```

Overriding:

- Whatever the Parent has by Default Available to the Child Class through Inheritance.
- If the Child Class is Not satisfied with the Parent Class Implementation then the Child is allowed to redefined that Method in the Child Class based on its Requirement.
- This Process is Called Overriding.
- The Parent Class Method which is Overridden is called *Overridden* Method and the Child Class Method which is Overriding is called *Overriding* Method.

Overridden Method

Overriding

Overriding Method

```
class P {
    public void property()
    {
        System.out.println ("Land + Gold + Cash");
    }
    public void mary()
    {
        System.out.println ("Subbu Lakshmi");
    }
}

class C extends P {
    public void mary()
    {
        System.out.println ("3sha/ 9tara/ 4me");
    }
}

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

            P p = new P();
            p.mary(); //Subbu Lakshmi (Parent Method)

            C c = new C();
            c.mary(); //3sha/ 9tara/ 4me (Child Method)

            P p1 = new C();
            p1.mary(); //3sha/ 9tara/ 4me (Child Method)

    }
}
```

- In Overriding Method Resolution Always Take Care by JVM based on Runtime Object.
- Hence Overriding is Considered as *Runtime Polymorphism* OR *Dynamic Polymorphism* OR *Late Binding*.
- The Process of Overriding Method Resolution is Also Known as *Dynamic Method Dispatch*.

Rules for Method Overriding:

- In Overriding, Method Names and Argument Types Must be Same. i.e Method Signatures Must be Same.
- In Overriding the Return Types Must be Matched. But this Rule is Applicable Only until 1.4 Version. From 1.5 Version onwards *Co- Variant Return Types* are allowed.

According to this Child Class Method Return Types Need Not to be Same as Parent Class Method Return Type. It's Child Types Also allowed.

```
class P {
    public Object m1() {
        return null;
    }
}
class C extends P {
    public String m1() {
        return null;
    }
}
```

Parent's Method Return Type	Object	Number	String	double
	✓	✓	×	×
Child Method Return Type	Object/String/StringBuffer…….	Number/Integer/Byte…….	Object	int

Note: Co-Varient Return Type Concept Applicable Only for Object Types but Not for Primitives.

- Private Methods are Not Visible in the Child Classes. Hence Overriding Concept is Not Applicable for Private Methods. But based on Our Requirement we can Define Exactly Same Private Method in Child Class. It is Valid but Not Overriding.

```
class P {
    private void m1() {}
}
class C extends P {
    private void m1() {}
}
```

It is Valid but Not Overriding

- We can't Override Parent Class final Methods in Child Class i.e., Overriding Concept is Not Applicable for final Methods.

```
class P {
    public final void m1() {}
}
class C extends P {
    public void m1() {}
}
```

CE: m1() in C cannot override m1() in P; overridden method is final

- We can Override Parent Class Abstract Method in Child Class to Provide Implementation.

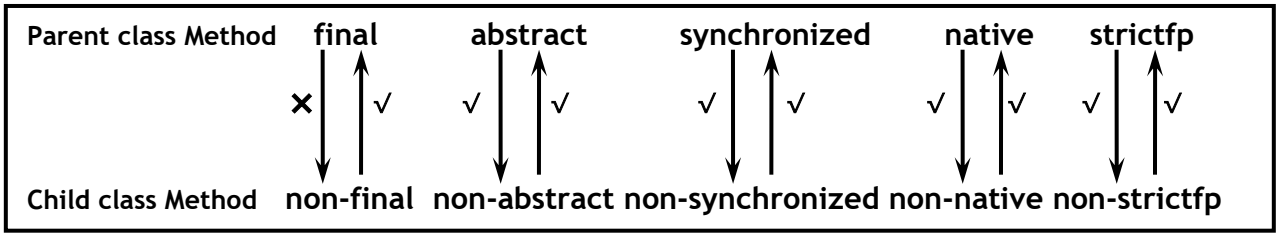
We can Override Parent Class Concrete Method as Abstract in Child Class.

```
class P {
        public void m1() {}
}
abstract class C extends P {
        public abstract void m1();
}
```

- Next Level Child is the Responsible to Provide Implementation.
- The Main Advantage of this Approach is we can Stop Availability of Parent Class Method Implementation to the Next Level Child Classes.

- The Following Modifiers won't Keep any Restrictions in Overriding.
  - synchronized,
  - native
  - strictfp

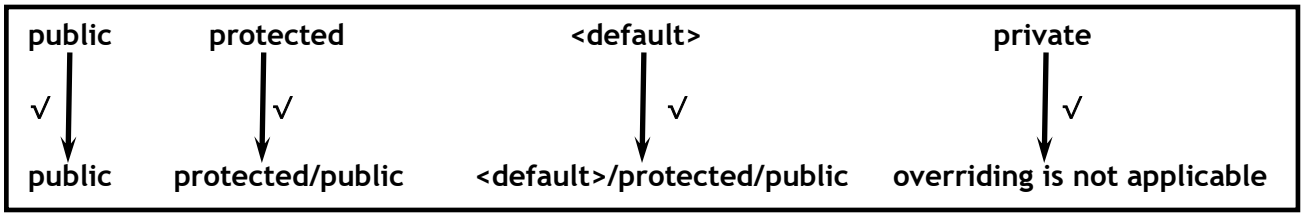
- The Following are Possible Types.



- While Overriding we can't Reduce Scope of Access Modifier. But we can Increase.

```
class P {
     public void m1() {}
}
class C extends P {
      protected void m1() {} // CE: m1() in C cannot override m1() in P
                                    attempting to assign weaker access privileges; was public
}
```

- The following is the List of Valid with Respect to Access Privileges.



private < default < protected < public

- If Child Class Method throws any Checked Exception Compulsory the Parent Class Method should throw the Same Checked Exception OR it's Parent. But there are No Restrictions for Un- Checked Exceptions.

```
import java.io.*;
class P {
      public void m1() throws IOException {}
}
class C extends P {
      public void m1() throws EOFException, InterruptedException{}
 // CE: m1() in C cannot override m1() in P
     overridden method does not throw InterruptedException
}
```

Examples:

1) P: public void m1() throws Exception
   C: public void m1 () {} √

2) P: public void m1()
   C: public void m1() throws Exception

   CE: m1() in C cannot override m1() in P
   overridden method does not throw Exception

3) P: public void m1() throws Exception
   C: public void m1() throws IOException √

4) P: public void m1() throws IOException
   C: public void m1() throws Exception

   CE: error: m1() in C cannot override m1() in P
   overridden method does not throw Exception

5) P: public void m1() throws IOException
   C: public void m1() throws EOFException, FileNotFoundException √

6) P: public void m1() throws IOException
   C: public void m1() throws EOFException, InterruptedException

   CE: error: m1() in C cannot override m1() in P
   overridden method does not throw InterruptedException

7) P: public void m1() throws IOException
   C: public void m1() throws EOFException, NullPointerException √

8) P: public void m1() throws IOException
   C: public void m1() throws ArithmeticException, NullPointerException, ClassCastException √

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Overriding wrt to Static Methods

- We can't Override a Static Method as Non- Static Otherwise we will get CE.

```
class P {
    public static void m1() {}
}
class C extends P {
    public void m1() {} //CE: m1() in C cannot override m1() in P
                                overridden method is static
}
```

Similarly we can't Override a Non- Static Method as Static.

```
class P {
    public void m1() {}
}
class C extends P {
    public static void m1() {}
    CE: m1() in C cannot override m1() in P
        overriding method is static
}
```

-

```
class P {
    public static void m1() {}
}
class C extends P {
    public static void m1() {}
}
```

It is Method Hiding but Not Overriding

It Seems Overriding Concept Applicable for Static Methods but it is Not Overriding. It is Method Hiding.

Method Hiding:

Method Hiding is Exactly Same as Overriding Except the following differences.

Method Hiding	Overriding
Both Parent Class and Child Class Methods should be Static.	Both Parent Class and Child Class Methods should be Non- Static.
Method Resolution always takes care by Compiler based on Reference Type.	Method Resolution always takes care by JVM based on Runtime Object.
Method Hiding is also known as *Compile Time Polymorphism/ Static Polymorphism/ Early Binding*.	Overriding is also known as *Runtime Polymorphism/ Dynamic Polymorphism/ Late Binding*.

Except these differences all Rules for Overriding and Method Hiding are Exactly Same.

```
class P {
      public static void m1() {
            System.out.println ("Parent Method");
      }
}
class C extends P {
       public static void m1() {
            System.out.println ("Child Method");
       }
}
class Test {
    public static void main(String arg[]) {
           P p = new P();
           p.m1(); //Parent Method
           C c = new C();
           c.m1(); //Child Method
           P p1 = new C();
           p1.m1(); //Parent Method
    }
}
```

- If Both Parent and Child Class Methods are Not- Static, then it will become Overriding. In this Case Output is
    - Parent Method
    - Child Method
    - Parent Method

Overriding wrt var-arg Method

- We can't Override var-arg Method with General Method, if we are trying to do it will become Overloading but not Overriding.
- We should Override var-arg Method with Another var-arg Method Only.

Overloading

```
class P {
    public void m1(int... i) {
        System.out.println ("Parent");
    }
}
class C extends P {
    public void m1(int i) {
        System.out.println ("Child");
    }
}
class Test {
public static void main(String args[]) {
        P p = new P();
        p.m1(); //Parent

        C c = new C();
        c.m1(10); //Child

        P p1 = new C();
        p1.m1(10); //Parent
    }
}
```

If we Replace Child Class Method also with var-arg Method then it will become Overriding. In this Case the Output is Parent, Child, and Child.

Overriding wrt Variables

- Overriding Concept is Applicable Only for Methods but Not for Variables.
- Variables Resolution always takes Care by Compiler, based on Reference Type (but not based on Run Time Object).
- This Rule is Same whether the Variable is Static OR Non- Static.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com


```
class P {
      int x = 888;
}
class C extends P {
      int x = 999;
}
class Test {
       public static void main(String[] args) {
                    P p = new P();
                    System.out.println(p.x); //888

                    C c = new C();
                    System.out.println(c.x);//999

                    P p1 = new C();
                    System.out.println(p1.x);//888
       }
}
```

Parent: non-static Child: non-static	static static	static non-static	non-static static
888	888	888	888
999	999	999	999
888	888	888	888

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Comparison between Overloading and Overriding

Properties	Overloading	Overriding
Method Names	Must be Same	Must be Same
Argument Types	Must be Different (at least Order)	Must be Same (Including Order)
Method Signature	Must be Different	Must be Same
private/final/static Methods	Can be Overloaded	Can't be Overridden
Return Types	No Restriction	Must be Same until 1.4 Version. But from 1.5 Version onwards co-variant return types are allowed.
Access Modifiers	No Restriction	The Scope of Access Modifier we can't reduce but we can increase.
throws clause	No Restriction	If Child Class Method throws any Checked Exception compulsory Parent Class Method should throw the Same Checked Exception OR it's Parent but no restrictions for Un-Checked Exceptions.
Method Resolution	Always takes care by Compiler based on Reference Type and Arguments.	Always takes care by JVM based on Runtime Object.
Other Names	Static Polymorphism OR Compile time Polymorphism OR Early Binding.	Runtime Polymorphism OR Dynamic Polymorphism OR Late Binding.

Note:

- In Overloading we have to Check Only Method Names (must be Same) and Argument Types (must be different).
- The remaining things we are not required to Check.
- But in Overriding we have to Check Everything Like Method Names, Argument Types, Return Types Etc.

Consider the following Method in Parent Class

public void m1(int i) throws IOException

In the Child Class which of the following Methods are allowed?

1) public void m1(int i) → Overriding

2) public static void m1(int i) → Overriding

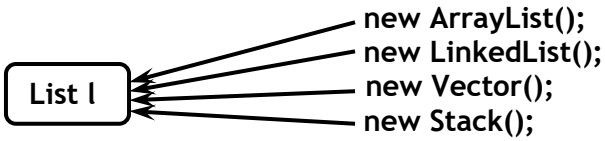
3) public final void m1(int i) throws Exception → Overriding
4) private static int m1(int i) throws Exception → Overloading

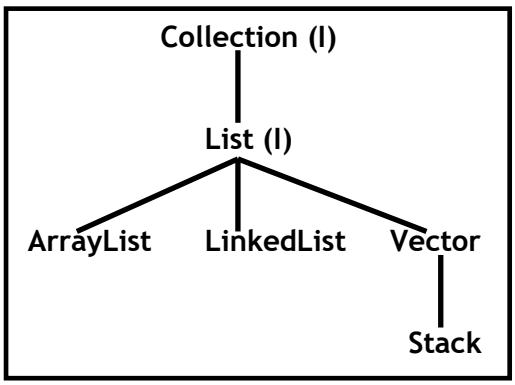
5) public static abstract void m1(double d)
CE: error: illegal combination of modifiers: abstract and static
: error: C is not abstract and does not override abstract method m1(double) in C

Polymorphism

Same Name but Multiple Forms is the Concept of Polymorphism.
Eg: We can Use the Same abs() for int, long and float Arguments.
abs (int);
abs (float);
abs (long);
are all Overloaded Methods

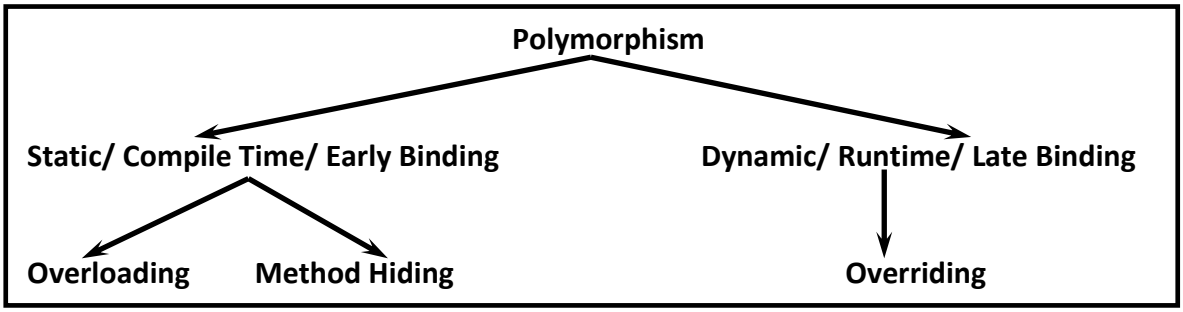
We can Use the Same List Reference to Hold any implemented Class Object.





What is the Difference between *ArrayList al = new ArrayList()*; and
List l = new ArrayList();

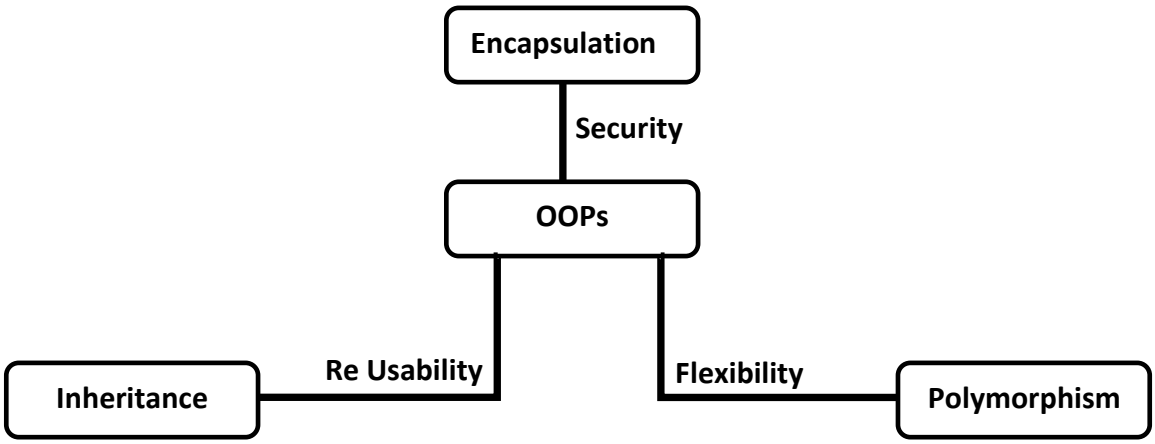
ArrayList al = new ArrayList(); (C c = new C();)	List l = new ArrayList(); (P p = new C();)
We can Use ArrayList Reference to Hold Only ArrayList Objects.	We can Use List Reference to Hold any implemented Class Objects.
If we Know the Type of Runtime Object at the beginning then we should Use this Approach.	If we don't Know the Type of Runtime Object at the beginning then we should Use this Approach.
On the Child Reference we can Call Both Parent Class and Child Class Methods.	On the Parent Reference we can Call Only Methods Available in Parent Class and Child Specific Methods we can't Call



Beautiful Definition of Polymorphism

A BOY Start LOVE with the Word FRIENDSHIP, but GIRL Ends LOVE with the Same Word FIRENDSHIP. Word is the Same but Attitude is different. This Beautiful Concept of OOPs is Nothing but Polymorphism.

Building Blocks of OOPs



DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Static Control Flow

```
class Base {
❶ static int i = 10; ❼

❷ static {
      m1(); ❽
      System.out.println("First Static Block"); ❿
  }
❸ public static void main(String[] args) {
      m1(); ⓭
      System.out.println("main()"); ⓯
  }
❹ public static void m1() {
      System.out.println(j); ❾ ⓮
  }
❺ static {
      System.out.println("Second Static Block"); ⓫
  }
❻ static int j = 20; ⓬
}
```

```
0
First Static Block
Secons Static Block
20
main()
```

```
i = 0; (RIWO)
J = 0 (RIWO)

i = 10 (R & W)
J = 20 (R & W)
```

Process of Static Control Flow:

Whenever we are executing a Java Class the following Sequence of Steps will be Performed Automatically.

❶ Identification of Static Members from Top to Bottom → (1-6) Steps.
i = 0 (RIWO) Read Indirect Write Only
j = 0 (RIWO)

❷ Execution of Static Variable Assignments and Static Blocks from Top to Bottom → (7-12) Steps.

i = 0 (R & W)
j = 0 (R & W)

❸ Execution of main() → (13-15) Steps.

Read In-directly Write Only Statement:

- If a Variable is Just Identified and Assign with Default Value then the Variable is Said to be in Read in-directly write Only State.
- If we are trying to Read any Variable Inside the Static Block that Read Operation is called Direct Read.
- If we are trying to Read a Variable Inside a Method that Operation is called In-direct Read.
- If a Variable is in RIWO State then we can't Perform Read Operation Directly Otherwise we will get CE: illegal forward reference.


```
class Test {
    ❶ static int x = 10;
    ❷ static {
        System.out.println(x);
    }
}
Output
10
RE: Exception in thread "main"
java.lang.NoSuchMethodError: main
```

x = 0 (RIWO)
x = 10

```
class Test {
    static {
        m1();
    }
    public static void m1() {
        System.out.println(i);
    }
    static int i = 10;
}
Output
0
RE: Exception in thread "main"
java.lang.NoSuchMethodError: main
```

```
class Test {
    static {
        System.out.println (x); //CE: illegal forward reference
    }
    static int x = 10;
}
```

Static Control Flow in Parent → Child:

Whenever we are executing Parent → Child Class the following Sequence of Events will be Performed Automatically.

- Identification of Static Members from Parent → Child [1-11]
- Execution of Static Variable Assignments and Static Blocks from Parent → Child. [12-22]
- Execution of Only Child Class main(). [23-25]

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com


```
class Base {
❶ static int i = 10;                    ⑫
❷ static {
        m1();                           ⑬
        SOP("Base First Static Block"); ⑮
   }
❸ public static void main(String[] args) {
        m1();
        SOP("Base main()");
   }
❹ public static void m1() {
        System.out.println(j);          ⑭
   }
❺ static int j = 20;                    ⑯
}
```

```
class Derived extends Base {
❻ static int x = 100;                   ⑰
❼ static {
        m2();                           ⑱
        SOP("Derived First Static Block"); ⑳
   }
❽ public static void main(String[] args) {
        m2();                           23
        SOP("Derived main()");          25
   }
❾ public static void m2() {
        System.out.println(y);          ⑲ 24
   }
❿ static {
        SOP("Derived Second Static Block"); 21
   }
⓫ static int y = 200;                   22
}
```

```
0
Base First Static Block
0
Derived First Static Block
Derived Second Static Block
200
Derived main()
```

1st Step: (1 - 11)
i = 0; (RIWO)
j = 0; (RIWO)
x = 0; (RIWO)
y = 0; (RIWO)

2nd Step: (12 - 22)
i = 10; (R & W)
j = 20; (R & W)
x = 100; (R & W)
y = 200; (R & W)

3rd Step: (23 - 25)
Execution of
Derived main()

Note: Whenever we are loading Child Class Automatically Parent Class will be loaded. But whenever we are loading Parent Class Child Class won't be loaded.

Static Block:

A Class can contain Static Block, it will be executed at the Time of Class loading Automatically. Hence while loading Class if we want to Perform any Activity we have to define that Inside this Static Block Only.

Eg: After loading Driver Class Compulsory we have to Register with DriverManager. But Every Database Driver Class contains a Static Block to Perform this Activity. Hence at the Time of Driver Class loading Only registering with DriverManager will be happen Automatically and we are Responsible to Perform Explicitly.

```
class Driver {
  static {
    //Register this Driver with DriverManager
  }
}
```

Eg: At the Time of Java Class loading the Corresponding Native Libraries should be loaded. Hence we have to define this Activity Inside Static Block.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

```
class Native {
   static {
      System.loadLibrary("Native Library Path:");
   }
}
```

Note: Inside a Class we can Take Any Number of Static Blocks and these Static Blocks will be executed from Top to Bottom.

Without using main() is it Possible to Print Some Statements to the Console?
Yes. By using Static Block.

```
class Test {
   static {
      System.out.println("Hello....I can Print ");
      System.exit(0);
   }
}
```

Without using main() and Static Block is it Possible to Print Some Statements to the Console?
Yes. There are Multiple Ways.

```
class Test {
   static int x = m1();
   public static  int m1() {
      System.out.println("Hello  I can Print");
      System.exit(0);
      return 10;
   }
}
```

```
class Test {
   static Test t = new Test();
   Test() {
      System.out.println("Hello I can Print");
      System.exit(0);
    }
}
```

```
class Test {
    static Test t = new Test();
    {
          System.out.println("Hello  I can Print");
          System.exit(0);
    }
}
```

Instance Block

Note: From 1.7 Version onwards to Run a Java Program main() is Mandatory. Even though we are writing Static Block, main() is Mandatory. Hence 1.7 Version onwards without writing main() it is not Possible to Print Some Statements to the Console.

Instance Control Flow:

- Whenever we are executing a Java Class 1st Static Control Flow will be executed.
- In the Static Control Flow whenever we are creating Object then Instance Control Flow will be executed.

```
class Parent {
    ❸ int i = 10;  ❾
    ❹ {
            m1(); ❿
            System.out.println("First Instance Block"); ⓬
      }
    ❺ Parent() {
            System.out.println("Constrctor");  ⓯
      }
    ❶ public static void main(String[] args) {
        ❷   Parent p = new Parent();
            System.out.println("Parent main()");
      }
    ❻ public void m1() {
        ⓫   System.out.println(j);
      }
    ❼ {
            System.out.println("Second Instance Block"); ⓭
      }
    ❽ int j = 20;  ⓮
}
```

```
0
First Instance Block
Second Instance Block
Constrctor
Parent main()
```

1st Step (3 - 8)
i = 0; (RIWO)
j = 0; (RIWO)

2nd Step (9 - 14)
i = 10; (R & W)
j = 20; (R & W)

Whenever we are creating an Object the following Sequence of Events will be performed Automatically.

- Identification of Instance Members from Top to Bottom (3 - 8).
- Execution of Instance Variable Assignments and Instance Blocks from Top to Bottom (9 - 14).
- Execution of Constructors (15)

Note:

- Static Control Flow is One Time Activity and that will Execute at the Time of Class loading.
- But Instance Control Flow is Not One Time Activity and that It will be Execute for Every Object Creation.

Instance Control Flow in Parent → Child Relation:

```
class Parent {
❹ int i = 10;                                   ⓯
❺ {
        m1(); ⓰
        SOP("Parent Instance Block"); ⓲
  }
❻ Parent() {
        SOP("Parent Constructor"); ⓴
  }
❶ public static void main(String[] args) {
        Parent P = new Parent();
        SOP("Parent main()");
  }
❼ public void m1() {
        System.out.println(j);  ⓱
  }
❽ int j = 20;                                   ⓳
}
```

```
class Child extends Parent {
❾ int x = 100;  21
❿ {
        m2();   22
        SOP("Child First Instance Block");24
  }
⓫ Child() {
        SOP("Child Constructor");  27
  }
❷ public static void main(String[] args) {
    ❸ Child c = new Child();
        SOP("Child main()");  28
  }
⓬ public void m2() {
        System.out.println(y); 23
  }
⓭ {
        SOP("Child Second Instance Block");  25
  }
⓮ int y =200;  26
}
```

```
0
Parent Instance Block
Parent Constructor
0
Child First Instance Block
Child Second Instance Block
Child Constructor
Child main()
```

Whenever we are creating Child Class Objects the following Sequence of Events will be Performed Automatically.

1) Identification of Instance Members from Parent to Child.(4 - 14)
   - i = 0 (RIWO)
   - j = 0 (RIWO)
   - x = 0 (RIWO)
   - y = 0 (RIWO)

2) Execution of Instance Variable Assignments and Instance Blocks only in Parent Class. (15 - 19)
   - i = 10 (R & W)
   - j = 20 (R & W)

3) Execution of Parent Class Constructor. 20 Step

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

4) Execution of Instance Variable Assignments and Instance Blocks in the Child Class. (21 - 26)

x = 100

y = 200

5) Execution of Child Class Constructor. 27TH Step

```
public class Initialization {
    private static String m1(String msg) {
            System.out.println (msg);
            return msg;
    }
    public Initialization() {
          m = m1("1");
    }
    {
          m = m1("2");
    }
    String m = m1("3");
    public static void main(String args[]) {
          Object obj = new Initialization();
    }
}
```

2 3 1

```
public class Initialization {
    private static String m1(String msg) {
          System.out.println (msg);
          return msg;
    }
    static String m = m1("1");
    {
           m = m1("2");
    }
    static {
         m = m1("3");
    }
    public static void main(String args[]) {
           Object obj = new Initialization();
    }
}
```

1 3 2

```
class Test {
     static {
        System.out.println ("First Static Block");
     }
     {
        System.out.println ("First Instance Block");
     }
     Test() {
         System.out.println ("Constructor");
     }
     public static void main (String args[]) {
         Test t = new Test();
     }
     static {
        System.out.println ("Second Static Block");
     }
     {
        System.out.println ("Second Instance Block");
     }
}
```

```
First Static Block
Second Static Block
First Instance Block
Second Instance Block
Constructor
```

From Static Area we can't Access Instance Members Directly because while executing Static Area JVM May Not Identify Instance Members.

```
class Test1 {
        int x = 10;
        public static void main(String[] args) {
                System.out.println(x);   }
        }
CE: error: non-static variable x cannot be referenced from a static context
```

Note: Object Creation is Most Costly Operation in Java. Hence if there is No Specific Requirement then it Never Recommended to Create Object.

In how Many Ways we can Create Object in Java? OR
How Many Ways we can get Object in Java?

We can Create Object in Java by using the following 5 Ways.

1) By using new Operator: Test t = new Test();

2) By using newInstance(); [By using Reflection API]

```
Test t = (Test)Class.forname("Test").newInstance();
                        OR
            (Test)Test.class.newInstance();
```

3) By using Factory Method:

```
Runtime r = Runtime.getRuntime();
DateFormat df = DF.getInstance();
```

4. By using clone():

```
Test t1 = new Test();
Test t2 = (Test)t1.clone();
```

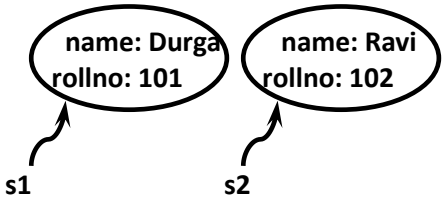
5. By using Deserialization:

```
FIS fis = new FIS("abc.ser");
OIS ois = new OIS(fis);
Test t2 = (Test)ois.readObject();
```

Constructor:

- Once we Create an Object Compulsory we should Perform Initialization. Then Only the Object is in a Position to Respond Properly.
- Whenever we are creating an Object Some Piece of Code will be executed Automatically to Perform Initialization of an Object. This Piece of the Code is Nothing but Constructor.
- Hence the Main Objective of Constructor is to Perform Initialization of an Object.

```
class Student {
      String name;
      int rollno;
      Student(String name, int rollno) {
          this.name = name;
          this.rollno = rollno;
      }
      public static void main(String[] args) {
          Student s1 = new Student("Durga",101);
          Student s2 = new Student("Ravi",102);
          ::::::::::::::::::::::::::::::::::::::::::::::::
          ::::::::::::::::::::::::::::::::::::::::::::::::
      }
}
```



Constructor Vs Instance Block

- The Main Purpose of Constructor is to Perform Initialization of an Object. Other than Initialization if we want to Perform any Activity for Every Object Creation then we should go for Instance Block.
- For Every Object Creation Both Constructor and Instance Block will be executed but 1st Instance Block followed by Constructor.
- Replacing Constructor with Instance Block OR Instance Block with Constructor is Not Possible. Both Concepts have their Own Purposes.

```
class Test {
   static int count = 0;

   {
       count++;
   }

   Test() {}

   Test(int i) {}

   public static void main(String[] args) {
         Test t1 = new Test();
         Test t2 = new Test(10);
         System.out.println ("The Number of Objects Created:" +count);
    }
}
```

```
The Number of Objects Created: 2
```

Rules for Constructors:

- The Name of the Constructor and the Name of the Class must be Same.
- Return Type Concept is Not applicable for Constructors Even void Also.
- By Mistake if we are trying to Declare Return Type then we won't get CE or RE. Simply it is treated as a Method.


```
class Test {
     void Test() { } → It is a Method but Not Constructor
}
```

- It is Legal (But Stupid) to have a Method whose Name is exactly Same as Class Name.

- The Only Applicable Modifiers for Constructor are public, private, protected and default. If we are trying to Apply any Other Modifier we will get CE.

```
class Test {
     final Test() { } // CE: modifier final not allowed here
}
```

Default Constructor:

- Every Class in Java contains Constructor.
- If we are Not writing any Constructor then the Compiler Always generates Default Constructor.
- If we are writing at Least One Constructor then the Compiler won't generate any Default Constructor.
- Hence Every Class contains either Programmer written Constructor OR Compiler generated Default Constructor but Not Both Simultaneously.

Prototype of Default Constructor:

- It is always no-arg Constructor.
- It contains only One Line *super()*; It is a no-arg Call to Super Class Constructor.
- The Modifier of the Default Constructor is Same as Class Modifier (But this Rule is applicable only for *public* and *default*)

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Programmers code	Compiler generated Code

class Test
{
}
	class Test
	{
	Test()
	{
	super();
	}
	}

public class Test
{
}
	public class Test
	{
	public Test()
	{
	super();
	}
	}

class Test
{
void Test()
{
}
}
	class Test
	{
	Test()
	{
	super();
	}
	void Test()
	{
	}
	}

class Test
{
Test()
{
}
}
	class Test
	{
	Test()
	{
	super();
	}
	}

class Test
{
Test(int i)
{
this();
}
Test()
{
}
}
	class Test
	{
	Test(int i)
	{
	this();
	}
	Test()
	{
	super();
	}
	}

class Test
{
Test(int i)
{
super();
}
}
	class Test
	{
	Test(int i)
	{
	super();
	}
	}

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

The 1st Line Inside Every Constructor should be either super OR this. If we are Not writing anything Compiler will always Place super().

Case1: We can use *super()* OR *this* only in 1st Line of the Constructor. If we are using anywhere else we will get CE.

```
class Test {
    Test() {
      System.out.println ("Constructor");
      super(); //CE: call to super must be first statement in constructor
    }
}
```

Case2: Within the Constructor we can use either super OR this but not Both Simultaneously.

```
class Test {
    Test() {
      super();
      this(); //CE: call to this must be first statement in constructor
    }
}
```

Case 3: We can Use *super()* and *this()* Only in Constructors. If we are using Outside of the Constructor we will get CE. That is we can Invoke a Constructor Directly from Another Constructor only.

```
class Test {
  public void m1() {
          super(); //CE: call to super must be first statement in constructor
  }
}
```

Differences between super() - this() and super - this:

super() - this()	super - this
These are Constructor Calls, to Call Super Class and Current Class Constructors.	These are Key Words to Refer Super Class and Current Class Instance Members.
We can write Only in Constructor as 1ST Statement Only.	Anywhere except Static Area.
We can write Only One but Not Both Simultaneously.	Any Number of Times.

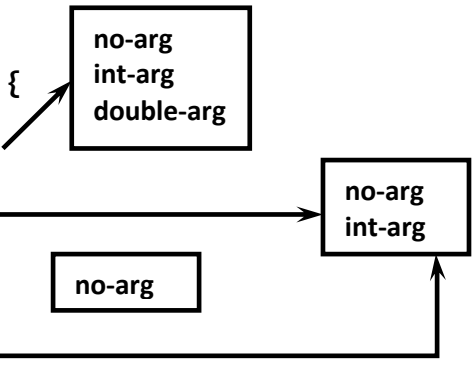
DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com


```
class Test {
    public static void main(String args[]) {
            System.out.println( super.hashCode());
         //CE: non-static variable super cannot be referenced from a static context
    }
}
```

Overloaded Constructor

A Class can contain Muiltiple Constructors with Same Name but different Argument Types. This Type of Constructors are called *Overloaded Constructors*. Hence Overloading Concept is applicable for Constructors.

```
class Test {

    Test(double d) {
          this(10);
          System.out.println("double-arg");
    }

    Test(int i) {
          this();
          System.out.println("int-arg");
    }

    Test() {
           System.out.println("no-arg");
    }

    public static void main(String arg[]) {
                Test t1 = new Test(10.8);
                Test t2 = new Test(10);
                Test t3 = new Test();
                Test t4 = new Test('a');
        }
}
```

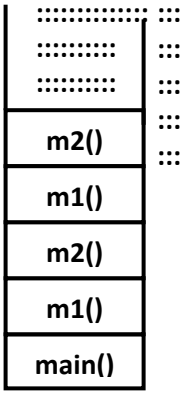


- Parent Class Constructors by Default won't Available to the Child and hence Inheritance Concept is Not Applicable for Constructors.
- As Inheritance Concept Not Applicable, by Default Overriding Concept Also Not Applicable.
- Every Class in Java including Abstract Class can contain Constructor. But Interface cannot have/ contain Constructor.

class Test { Test() {} } √	abstract class Test { Test() {} } √	interface Test { Test() {} } ×

Case 1: Recursive Method Call is always Runtime Exception saying StackOverfolwError. But in Our Program if there is any Chance of Recursive Constructor Invocation we will get CE i.e., Recursive Method Invocation is the RE where as Recursive Constructor is a CE.

```
class Test {
     public static void m1() {
            m2();
     }
     public static void m2() {
            m1();
     }
     public static void main(String arg[]) {
            m1();
            System.out.println("Hello.....Hai");
       // RE: Exception in thread "main" java.lang.StackOverflowError
     }
}
```



Run Time Flow

```
class Test {
    Test() {
        this(10);
    }
    Test(int i) {  //CE: recursive constructor invocation
            this();
    }
    public static void main(String arg[]) {
            System.out.println("Hello.....Hai");
   }
}
```

Case 2:

- If the Parent Class contains Argument Constructor then while writing Child classes we have to Take Special Care with Respect to Constructors.
- Whenever we are writing any Argument Constructor it is Highly Recommended to write no-arg Constructor Also.


```
class P {}
class C extends P {} ✓
```

```
class P {
    P() {}
}
class C extends P {} ✓
```

```
class P {
    P(int i) {}
}
class C extends P {} ✗

CE: constructor P in class P cannot be applied to given types;
    required: int
    found: no arguments
    reason: actual and formal argument lists differ in length
```

Case 3: If the Parent Class Constructor *throws* Some Checked Exception. Compulsory the Child Class Constructor should throw the Same Checked Exception OR its Parent Otherwise CE.

```
import java.io.*;
class P {
   P() throws IOException {}
}
class C extends P {} //CE: unreported exception IOException in default constructor
```

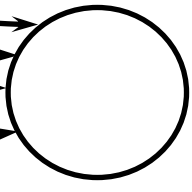
```
import java.io.*;
class P {
    P() throws IOException {}
}
class C extends P {
     C() throws IOException|Exception|Throwable {
          super();
     }
}
```

Singleton Classes: For any Java Class if we are allow to Create Only One Object Such Type of Class is called Singleton Class.
Eg: Runtime, ActionServlet, BusinessDelegate, ServiceLocator etc.

Advantage:

- Instead of creating a Separate Object for Every Requirement we can Create Only One Object and we can Reuse the Same Object for Every Similar Requirement.
- So that *Memory Utilization* and *Performance* will be Improved.
- This is the Main Advantage of Singleton Classes.

```
Runtime r1 = Runtime.get Runtime ();        r1
Runtime r2 = Runtime.get Runtime ();        r2
Runtime r3 = Runtime.getRuntime();          r3
      ::                                    ::
      ::                                    ::
Runtime r1000 = Runtime.getRuntime ();      r1000
```



DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Note:

- We can't Create Singleton Class Objects by using Constructor.
- We can Create by using Factory Method.

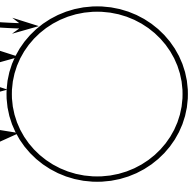
Creation of Our Own Singleton Classes:

We can Create Our Own Singleton Classes; for this we have to Use

- Private Constructor,
- Factory Method,
- One Private Static Variable

```
class Test {
    private static Test t = null;

    private Test() {}

    private static Test getTest() {
        if(t == null) {
            t = new Test();
        }
        return t;
    }

    public Object clone() {
        return this;
    }
}
```

```
Test t1 = Test.getTest();          t1
Test t2 = Test.getTest();          t2
Test t3 = Test.getRuntime();       t3
      ::                           ::
      ::                           ::
Test t1000 = Test.getTest();       t1000
```



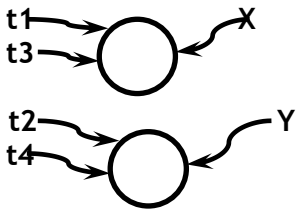
In the Above Example for Test Class we are allowed to Create Only One Object and Hence it is Singleton Class.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Creation of Our Own Doubleton Classes:

```
class Test {
        private static Test x = null;

        private static Test y = null;

        private Test() {}

        public static Test getTest() {
                if(x == null) {
                        x = new Test();
                        return x;
                }
                else if(y == null) {
                        y = new Test();
                        return y;
                }
                else if(Math.random(1 < 0.5)) return x;
                else return y;
        }
}
```

```
Test t1 = Test.getTest();
Test t2 = Test.getTest();
        ::
        ::
```



Similarly we can Create Thribleton, Tenton any Xxxton Classes.

Note: This Type of Approach we can Use while developing Connection Pools, Thread Pools etc.

Class is Not final but we are Not allowed to Create Child Class. How it is Possible?

If we Declare Every Constructor as Private then we can't Create Child Class for that.

```
class Parent {
    private Parent() {}
}
class Child extends Parent {} //CE: Parent() has private access in Parent
```

Coupling: The Degree of Dependency between the Components is Called Coupling.

Eg:

```
class A {
    int i = B.j;
}
```

```
class B {
    static int j = C.m1();
}
```

```
class C {
    public static int m1() {
        return D.k;
    }
}
```

```
class D {
    static int k= 10;
}
```

The Dependency between the Above Classes is More and Hence these Components are Said to be Tightly Coupled with Each Other.

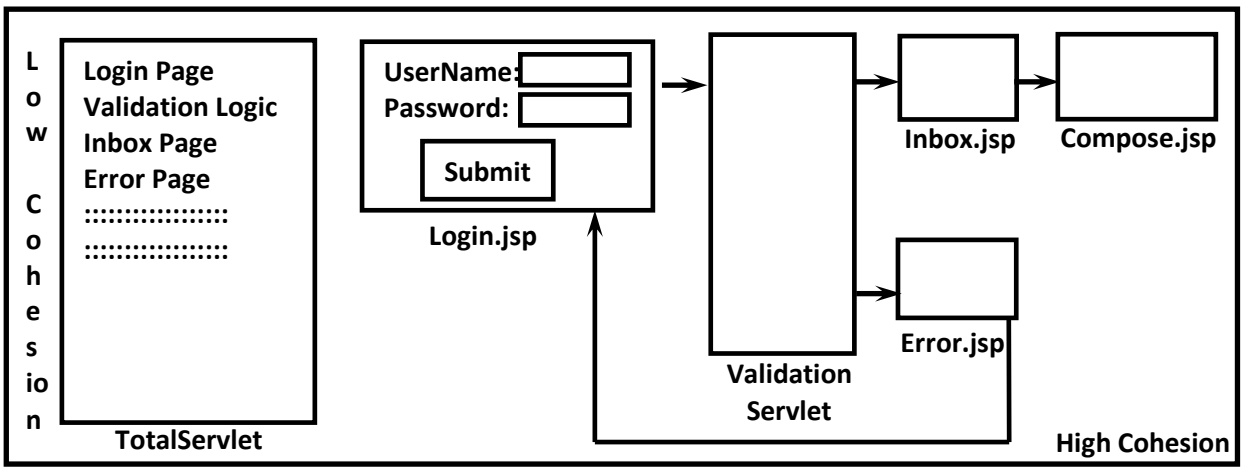
Tightly Coupling is Not a Good Programming Practice. Because it has Several Serious Disadvantages.

- Without effecting remaining Components we can't Modify any Component. Hence Enhancement will become Difficult.
- It doesn't Promote Re-usability of the Code.
- It Reduces Maintainability of the Application.

Hence we have to Maintain Dependency between the Components as Less as Possible. That is Loosely Coupling is Always Good Programming Practice.

Cohesion:

For Every Component we have to Define a Clear Well defined Functionality. Such Type of Component is Said to be follow High Cohesion.



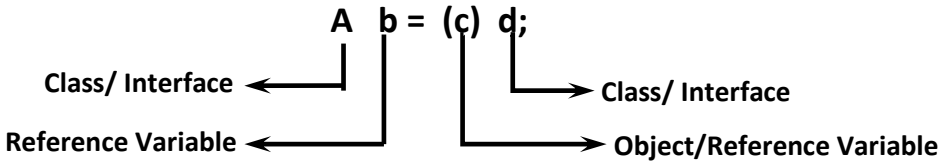
High Cohesion is Always Good Programming Practice. Because it has Several Advantages.

- Without effecting remaining Components we can Modify any Component. Hence Enhancement will become Very Easy.
- It Promotes Re- Usability of the Code (wherever Validation Logic is required we can Re- Use the Same ValidateServlet without re-writing).
- It Improves Maintainability of the Application.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Note: Loosely Coupling and High Cohesion are Always Good Programming Practices.

Type Casting



Compile Time Checking:

❶ The Type of d and c Must have Some Relation.
(Either Parent → Child OR Child → Parent OR Same Type) Otherwise we will get CE.

```
Object o = new String ("Durga");
StringBuffer sb = (StringBuffer)o; √
```

```
String s = new String ("Durga");
StringBuffer sb = (StringBuffer)s;

CE: inconvertible types
     required: StringBuffer
     found:    String
```

❷ 'C' should be Either Same OR Derived Type of A. Otherwise we will get CE.

```
String s = new String ("Durga");
StringBuffer sb = (StringBuffer)s;

CE: inconvertible types
     required: StringBuffer
     found:    String
```

Runtime Checking

The Runtime Object Type of 'd' Must be either Same OR derived Type of 'C'.
Otherwise we will get Runtime Exception saying ClassCastException : 'd' type

```
Object o = new String("Durga") ;
StringBuffer sb = (StringBuffer)o;

//RE: Exception in thread "main" java.lang.ClassCastException:
java.lang.String cannot be cast to java.lang.StringBuffer
```

```
Object o = new StringBuffer("Ravi") ;
StringBuffer sb = (StringBuffer)o;  √
```

❖ Strictly speaking through Type Casting we are not creating any New Object for the existing Object. We are providing another Type of Reference Variable.

```
String s = new String("Durga");
Object o = (Object)s;
System.out.println(s == o); //true
```

String s, Object o → Durga

```
class Test {
    public static void main(String args[]) {
        Integer i = new Integer(10);
        Number n = (Number)i;
        Object o = (Object)n;
        System.out.println(i.hashCode()); //10
        System.out.println(n.hashCode()); //10
        System.out.println(o.hashCode()); //10
        System.out.println(i == n); //true
        System.out.println(i == o); //true
    }
}
```

Integer i, Number n, Object o → 10

Examples:

```
C c = new C();
```

P → m1()
C → m2()

c.m1() √

c.m2() √

((P)c).m1() √

((P)c).m2()
CE: cannot find symbol
found: method m2()
location: class P

```
C c = new C(c);
```

c.m1(); // C

((B)c).m1(); //C

((A) ((B)c)).m1(); //C

A → m1() { sopln ("A"); }
B → m1() { sopln ("B"); }
C → m1() { sopln ("C"); }

It is Overriding and Method Resolution is Always based on Runtime Object.

```
C c = new C(c);
```

c.m1(); // C

((B)c).m1(); //B

((A) ((B)c)).m1(); //A

A → static m1() { sopln("A"); }
B → static m1() { sopln("B"); }
C → static m1() { sopln("C"); }

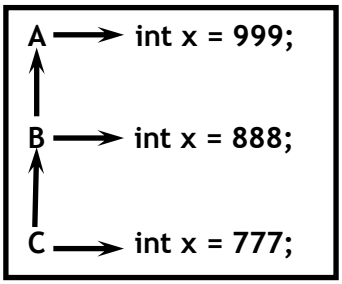
It is Method Hiding and Method Resolution is Always based on Reference Type.

C c = new C();

Sopln(c.x); //777

Sopln((B)c).x); //888

sopln((A) ((B)c)).x); //999



Overriding Concept is Not Applicable for Variables and Variable Resolution is Always Takes Care by Compiler On Reference Type Based

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

OOPS Practice Questions

Exam Objectives:

1. Compare and contrast the features and components of Java such as: object orientation, encapsulation, etc.
2. Describe inheritance and its benefits
3. Create methods with arguments and return values; including overloaded methods
4. Apply encapsulation principles to a class
5. Develop code that makes use of polymorphism; develop code that overrides methods; differentiate between the type of a reference and the type of an object
6. Create and overload constructors; differentiate between default and user defined constructors
7. Use super and this to access objects and constructors
8. Determine when casting is necessary

Q1. Which three statements describe the object oriented featues of the java language?

A. Objects cannot be reused
B. A sub class can inherit from a super class
C. Objects can share behaviours with other objects
D. A package must contains more than one class
E. Object is the root class of all other objects
F. A main method must be declared in every class

Answer: B,C,E

Q2. What is the name of the Java concept that uses access modifiers to protect variables and hide them within a class?

A. Encapsulation
B. Inheritance
C. Abstraction
D. Instantiation
E. Polymorphism

Ans: A

Explanation: This concept is data hiding, but that option is not available and hence we can choose encapsulation

Q3. Which statement best describes encapsulation?

A. Encapsulation ensures that classes can be designed so that only certain fields and methods of an object are accessible from other objects

B. Encapsulation enures that classes can be designed so that thier methods are inheritable

C. Encapsulation ensures that classes can be designed with some fields and methods declared as abstract.

D. Encapsulation ensures that classes can be designed so that if a method has an argument X,any subclass of X can be passed to that methods.

Answer: A

Q4. Given the following two classes:

```
1)  public class Customer
2)  {
3)     ElectricAccount acct=new ElectricAccount();
4)     public void useElectricity(double kwh)
5)     {
6)        acct.addKwh(kwh);
7)     }
8)  }
9)  public class ElectricAccount
10) {
11)    private double kwh;
12)    public double rate=0.09;
13)    private double bill;
14)    //Line-1
15) }
```

How should you write methods in ElectricAccount class at Line-1 so that the member variable bill is always equal to the value of the member variable kwh multiplied by the member variable rate?

Any amount of electricity used by Customer(represented by an instance of the Customer class) must contribute to the Customer's bill(represented by member variable bill) through the method useElectricity() method. An instance of the customer class should never be able to tamper with or decrease the value of the member variable bill?

A)

```
1)  public void addKwh(double kwh)
2)  {
3)     this.kwh+=kwh;
4)     this.bill=this.kwh*this.rate;
5)  }
```

B)

```
1)  public void addKwh(double kwh)
2)  {
3)     if(kwh>0)
4)     {
5)        this.kwh+=kwh;
6)        this.bill=this.kwh*this.rate;
7)     }
8)  }
```

C)

```
1)  private void addKwh(double kwh)
2)  {
3)     if(kwh>0)
4)     {
5)        this.kwh+=kwh;
6)        this.bill=this.kwh*this.rate;
7)     }
8)  }
```

D)

```
1)  public void addKwh(double kwh)
2)  {
3)     if(kwh>0)
4)     {
5)        this.kwh+=kwh;
6)        setBill(this.kwh)
7)     }
8)  }
9)  public void setBill(double kwh)
10) {
11)    bill=kwh*rate;
12) }
```

Answer: C

Explanation:

Any amount of electricity used by Customer(represented by an instance of the Customer class) must contribute to the Customer's bill(represented by member variable bill) through the method useElectricity() method.

means no one is allowed to call addKwh() method directly, shoulbe be private

An instance of the customer class should never be able to tamper with or decrease the value of the member variable bill

if we pass -ve value for kwh then we can decrease the bill. To prevent this we should check kwh>0

Q5. Given the following code fragment:

```
1)  public class Rectangle
2)  {
3)     private double length;
4)     private double height;
5)     private double area;
6)     public void setLength(double length)
7)     {
8)        this.length=length;
9)     }
10)    public void setHeight(double height)
11)    {
12)       this.height=height;
13)    }
14)    public void setArea()
15)    {
16)       area=length*height;
17)    }
18) }
```

Which two changes would encapsulation this class and ensure that the area field is always equal to length*height,whenever Rectangle class is used?

A. Change the area field to public
B. Change the setArea() method to private?
C. Call the setArea() method at the beginning of the setLength() method
D. Call the setArea() method at the end of the setLength() method
E. Call the setArea() method at the beginning of the setHeight() method
F. Call the setArea() method at the end of the setHeight() method

Answer: B,F

Q6. Given the following classes:

```
1)  public class Employee
2)  {
3)     public int salary;
4)  }
5)  public class Manager extends Employee
6)  {
7)     public int budget;
8)  }
9)  public class Director extends Manager
10) {
11)    public int stockOptions;
```

```
12) }
```

And given the following main method:

```
1)  public static void main(String[] args)
2)  {
3)     Employee e = new Employee();
4)     Manager m = new Manager();
5)     Director d = new Director();
6)     //Line 1
7)  }
```

Which two options fail to compile when placed at Line 1 of the main method?

A. e.salary=50_000;
B. d.salary=80_000;
C. e.budget=2_00_000;
D. m.budget=1_00_000;
E. m.stockOption=500;
F. d.stockOption=1_000;

Answer: C and E

Q7.Given the code fragment:

```
1)  abstract class Parent
2)  {
3)     protected void resolve()//Line-1
4)     {
5)     }
6)     abstract void rotate();//Line-2
7)  }
8)  class Child extends Parent
9)  {
10)    void resolve()//Line-3
11)    {
12)    }
13)    protected void rotate()//Line-4
14)    {
15)    }
16) }
```

Which two modifications,made independently, enable the code to compile?

A. Make that method at Line-1 public
B. Make that method at Line-2 public
C. Make that method at Line-3 public
D. Make that method at Line-3 protected

E. Make that method at Line-4 public

Answer: C,D

Explanation: While overriding, we cannot reduce the scope of access modifier, but we can increase the scope

Q8. Given:
Base.java:

```
1)  class Base
2)  {
3)     public void test()
4)     {
5)        System.out.println("Base");
6)     }
7)  }
```

DerivedA.java:

```
1)  class DerivedA extends Base
2)  {
3)     public void test()
4)     {
5)        System.out.println("DerivedA");
6)     }
7)  }
```

DerivedB.java

```
1)  class DerivedB extends DerivedA
2)  {
3)     public void test()
4)     {
5)        System.out.println("DerivedB");
6)     }
7)     public static void main(String[] args)
8)     {
9)        Base b1= new DerivedB();
10)       Base b2= new DerivedA();
11)       Base b3= new DerivedB();
12)       b1=(Base)b3;
13)       Base b4=(DerivedA)b3;
14)       b1.test();
15)       b4.test();
16)    }
17)
```

```
18) }
```

What is the result?

A.
Base
DerivedA

B.
Base
DerivedB

C.
DerivedB
DerivedB

D.
DerivedB
DerivedA

E. A ClassCastException is thrown at runtime

Answer: C

Q9. Which two are benefits of polymorphism?

A. Faster Code at Runtime
B. More efficient Code at Runtime
C. More Dynamic Code at Runtime
D. More Flexible and Reusable Code at Runtime
E. Code that is protected from extension by other classes

Answer: C,D

Q10. Which three statements are true about the structure of a Java class?

A) public class should compulsory contains main method
B) A class can have only one private constructor
C) A method can have the same name as variable
D) A class can have overloaded static methods
E) The methods are mandatory components of a class
F) The fields need not be initialized before use.

Answer: C, D,F

Q11. Given:

```
1)  class A
2)  {
3)     public void test()
4)     {
5)        System.out.print("A");
6)     }
7)  }
8)  class B extends A
9)  {
10)    public void test()
11)    {
12)       System.out.print("B");
13)    }
14) }
15) public class C extends A
16) {
17)    public void test()
18)    {
19)       System.out.print("C");
20)    }
21)    public static void main(String[] args)
22)    {
23)       A a1 = new A();
24)       A a2 = new B();
25)       A a3 = new C();
26)       a1.test();
27)       a2.test();
28)       a3.test();
29)    }
30) }
```

What is the output?

A. AAA
B. ABC
C. AAB
D. ABA

Ans: B

Q12. Given:

```
1)  public class Test
2)  {
3)     public static void sum(Integer x,Integer y)
4)     {
```

```
5)      System.out.println("Integer sum is:"+(x+y));
6)    }
7)    public static void sum(double x,double y)
8)    {
9)      System.out.println("double sum is:"+(x+y));
10)   }
11)   public static void sum(float x,float y)
12)   {
13)     System.out.println("float sum is:"+(x+y));
14)   }
15)   public static void sum(int x,int y)
16)   {
17)     System.out.println("int sum is:"+(x+y));
18)   }
19)   public static void main(String[] args)
20)   {
21)     sum(10,20);
22)     sum(10.0,20.0);
23)   }
24) }
```

What is the result?

A.
int sum is:30
double sum is:30.0

B.
int sum is:30
float sum is:30.0

C.
Integer sum is:30
double sum is:30.0

D.
Integer sum is:30
float sum is:30.0

Answer: A

Q13. Given the code

```
1)  public class Test
2)  {
3)    public static void main(String[] args)
4)    {
5)      Short s1=200;
6)      Integer s2=400;
7)      Long  s3=(long)s1+s2;  //Line-1
8)      String s4=(String)(s3*s2);// Line-2
9)      System.out.println(s3);
```


```
10)    }
11) }
```

What is the result?

A. 600
B. Compilation Fails at Line-1
C. Compilation Fails at Line-2
D. A ClassCastException is thrown at Line-1
E. A ClassCastException is thrown at Line-2

Answer: C

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com