# Day-8 JA-111 Batch

## Topics

- **Super class reference variable and child class object**
- **Dynamic Method Dispatch (Dynamic Binding)**
- **Up-casting & Down-casting for reference types**
- **Overriding methods of Object class**

## Super class reference variable and child class object

- An object of subclass has everything that an object of super class has so object of subclass can be considered as object of super class.
- A reference variable of super class can point to the object of child class but using that reference variable we can access only those members that are defined in super class because reference variable of super class do not have any knowledge about the members defined in subclass.

**An Example**

```
package com.masai.sprint2;
class A{
  int i;
  int j;

  void setA(int a, int b){
    i = a;
    j = b;
  }

  void showA(){
    System.out.println("i = " + i);
    System.out.println("j = " + j);
  }
}

package com.masai.sprint2;
class B extends A{
  int k;

  void setB(int c){
    k = c;
  }

  void showB(){
    System.out.println("k = " + k);
  }

  public static void main(String... args){
    A a = new A();  //Ref: A object: A
    a.setA(10, 20);
    a.showA();
    System.out.println();
    B b = new B();  //Ref: B object: B
```

```
    b.setA(100, 200);
    b.setB(300);
    b.showA();
    b.showB();
    System.out.println();
    A a1 = new B();  //Ref: A object: B
    a1.setA(1000, 2000);
    a1.showA();
    //a1.setB(3000);  Error
    //a1.showB();

    //B b1 = new A();  Error, the sub class reference variable cannot point to
object of super class
  }
}
```

Output

```
i = 10
j = 20

i = 100
j = 200
k = 300

i = 1000
j = 2000
```

## Dynamic Method Dispatch (Dynamic Binding)

- Using method overriding, we are redefining the behaviour of general class into the specialized class i.e. it is used to define the general behaviour of a super class in the sub class according the specialized need.
- A super class ref. variable can point to child class object and this ref variable can access methods defined in super class only. The overridden method is defined in super class and sub class both so the overridden method can be called using ref variable of super class such that-
  a. If reference variable is of superclass and object is of super class then version defined in super class will be called.
  b. If reference variable is of superclass and object is of sub class then version defined in sub class will be called.
- The calling decision of overridden method is taken at run time that's why it is a form of run time polymorphism. The concept of calling the overridden method at runtime is called run time polymorphism.

An Example

```
package com.masai.sprint2;
public class Cat{
  void sound(){
    System.out.println("Meow...");
  }
}

package com.masai.sprint2;
public class Tiger extends Cat{
```

```
  void sound(){
    System.out.println("Roar...");
  }

  public static void main(String args[]){
    Cat c = new Cat();       //Ref.Var: Super class Object: Super class
    c.sound();
    Tiger t = new Tiger();   //Ref.Var: Sub class Object: sub class
    t.sound();

    c = t;  //Ref. Var.: Super class Object: sub class
    c.sound();
  }
}
```

Output:

```
Meow...
Roar...
Roar...
```

Another Example

```
package com.masai.sprint2;
public class EnggStudent {
  private double tuitionFee;
  private double cautionMoney;

  public EnggStudent(double tuitionFee, double cautionMoney) {
    this.tuitionFee = tuitionFee;
    this.cautionMoney = cautionMoney;
  }

  public double getTuitionFee() {
    return tuitionFee;
  }

  public void setTuitionFee(double tuitionFee) {
    this.tuitionFee = tuitionFee;
  }

  public double getCautionMoney() {
    return cautionMoney;
  }

  public void setCautionMoney(double cautionMoney) {
    this.cautionMoney = cautionMoney;
  }

  public double getTotalFee() {
    return (tuitionFee + cautionMoney);
  }
}

package com.masai.sprint2;
```

```
public class HostelStudent extends EnggStudent{
  private double hostelFee;
  private double hostelCautionMoney;

  public HostelStudent(double tuitionFee, double cautionMoney, double
hostelFee, double hostelCautionMoney) {
    super(tuitionFee, cautionMoney);
    this.hostelFee = hostelFee;
    this.hostelCautionMoney = hostelCautionMoney;
  }

  public double getHostelFee() {
    return hostelFee;
  }

  public void setHostelFee(double hostelFee) {
    this.hostelFee = hostelFee;
  }

  public double getHostelCautionMoney() {
    return hostelCautionMoney;
  }

  public void setHostelCautionMoney(double hostelCautionMoney) {
    this.hostelCautionMoney = hostelCautionMoney;
  }

  @Override
  public double getTotalFee() {
    return super.getTuitionFee() + (hostelFee + hostelCautionMoney);
  }
}

package com.masai.sprint2;
public class DayScholar extends EnggStudent{
  private double busFee;

  public DayScholar(double tuitionFee, double cautionMoney, double busFee) {
    super(tuitionFee, cautionMoney);
    this.busFee = busFee;
  }

  public double getBusFee() {
    return busFee;
  }

  public void setBusFee(double busFee) {
    this.busFee = busFee;
  }

  @Override
  public double getTotalFee() {
    return super.getTotalFee() + busFee;
```

```
  }
}

package com.masai.sprint2;
public class TesterDMD {
  public static void main(String[] args) {
    EnggStudent es = new EnggStudent(70000, 7500);
    System.out.println("The total fee paid by student is " + es.getTotalFee());

    es = new HostelStudent(70000, 7500, 65000, 6500);
    System.out.println("The total fee paid by hosteller student is " +
es.getTotalFee());

    es = new DayScholar(70000, 7500, 13500);
    System.out.println("The total fee paid by Day scholar student is " +
es.getTotalFee());
  }
}
```

Output

```
The total fee paid by student is 77500.0
The total fee paid by hosteller student is 141500.0
The total fee paid by Day scholar student is 91000.0
```

***We Activity***

Q.1: What is output of following code-

```
class A{
  public void show(){
    System.out.println("A");
  }
}

class B extends A{
  void show(){
    System.out.println("B");
  }

  public static void main(String args[]){
    A a = new B();
    a.show();
  }
}
```

A. Will print A
B. Will print B
C. Will print A & B
D. None of these

Q.2: What is output of following code

```
class A{
  private void foo(){
    System.out.println("A");
  }
```

```
}

class B extends A{
  public void foo(){
    System.out.println("B");
  }

  public static void main(String args[]){
    A a = new B();
    a.foo();
  }
}
```

A. Will print A
B. Will print B
C. Will print A & B
D. None of these

Q.3 What is output of following code

```
class A{
  static void fun(){
    System.out.println("A");
  }
}
class B extends A{
  static void fun(){
    System.out.println("B");
  }

  public static void main(String args[]){
    A a = new B();
    a.fun();
  }
}
```

A. Will print A
B. Will print B
C. Will print A & B
D. None of these

Q.4: What is output of following code-

```
class P{}
class Q extends P{}

class A{
  Q show(){
    System.out.println("A");
    return new Q();
  }
}
class B extends A{
  P show(){
    System.out.println("B");
    return new P();
  }
```

```
  public static void main(String args[]){
    A a = new B();
    a.show();
  }
}
```

A. Will print A
B. Will print B
C. Will print A & B
D. None of these

Q.5 What is output of following code-

```
class P{}
class Q extends P{}

class A{
 P show(){
   System.out.println("A");
   return new P();
 }
}

class B extends A{
 Q show(){
   System.out.println("B");
   return new Q();
 }
}

class Demo{
 public static void main(String args[]){
   A a = new A();
   P p = a.show();

   a = new B();
   P p1 = a.show();
 }
}
```

A. Will print A
B. Will print B
C. Will print A & B
D. None of these

Q.1 RIGHT ANSWER: D

When overriding a method in subclass make sure to use same or less restrictive access modifier, never use more restrictive access modifier because you cannot downgrade the access modifier but you can upgrade.

private (Most restrictive) < default (no modifier) < protected < public (Least Restrictive)

Q.2 RIGHT ANSWER: D

Explanation: private methods are not accessiable outside the class so they cannot be overridden. so calling a private method outside class is an error.

Q.3 RIGHT ANSWER: A
Explanation: Run time polymorphism is not applicable for the static method because they can be caller using class name so calling of static method is always resolved at compile time so calling will be performed on behalf of reference name/class-name.

Q.4 RIGHT ANSWER: D
Explanation: Overridden method can return same or subtype but it cannot return super type. This rule is called covariant return type.

Q.5 RIGHT ANSWER: C

**instanceof operator**

- It is used to check if the object belong to the specified class or not
- Syntax: obj-name instanceof class-name

```
class A{}
class B extends A{}

class Demo{
  public static void main(String args[]){
    A a = new A();
    System.out.println("a instanceof A? " + (a instanceof A));  //true
    System.out.println("a instanceof B? " + (a instanceof B));  //false
    System.out.println();

    B b = new B();
    System.out.println("b instanceof B? " + (b instanceof B));  //true
    System.out.println("b instanceof A? " + (b instanceof A));  //true
    System.out.println();

    A temp = new B();
    System.out.println("temp instanceof A? " + (temp instanceof A));  //true
    System.out.println("temp instanceof B? " + (temp instanceof B));  //true
  }
}
```

## Up-casting & Down-casting for reference types

```
class S{}
class T extends S{}

S sOne = new S();  //Ref Var: S Object: S No Casting
T tOne = new T();  //Ref Var: T Object: T No Casting

S sTwo = new T();  //Ref Var: S Object: T Up-casting i.e. reference variable: super class and object: sub class. It is automatic.
T tTwo = sTwo;  //Assigning a super type to the sub type i.e. Down-casting... it is not automatic so error

T tTwo = (T)sTwo;  //Assigning a super type to the sub type i.e. Down-casting... It is manual so no issue at the compile time and after this assignment you will find one more thing that the reference variable tTwo of the class T will point to the object of class T so at run time there will be no
```

```
error.
S sThree = new S();  //Ref Var: S Object: S No Casting

T tThree = sThree;    //Assigning a super type to the sub type i.e. Downcasting... it is not automatic so error

T tThree = (T)sThree;  //Assigning a super type to the sub type i.e. Down-casting. It is manual so no issue at the compile time but at run time system JVM will found that sThree is pointing to the object of class S and after this assignment, the reference variable of class T that is tThree points to object of class S which is not allowed. So JVM will produce a Run time error named ClassCastException.
```

## Overriding methods of Object class

In java we do not have operator overloading so meaning of == cannot be redefined; if you wanted to check if two different references point to same object then use == but method overriding is available so meaning of equals() can be redefined; if you wanted to check if two objects has same content (may be for all fields or for one field) then override equals method and then use it.

- The default implementation of equals method uses == so if two references point to same memory location then return true and hashcode for both references will be same. So when override equals method make sure to override hashcode also such that for two objects if equals method return true then their hashcode must be same.
- If you are overriding the hashCode method then it is not required to override equals() method because for two different objects the generated hashcode may be same.

The above both points are convention not rule so if you do not follow then no error will be there but for long run/correct implementation follow both because you see its impact in collection framework.

```
package com.masai.sprint2;
class Car{
  private String registratrionNumber;
  private double price;
  private int noOfSeats;

  public Car(String registratrionNumber, double price, int noOfSeats) {
    this.registratrionNumber = registratrionNumber;
    this.price = price;
    this.noOfSeats = noOfSeats;
  }

  public String getRegistratrionNumber() {
    return registratrionNumber;
  }

  public void setRegistratrionNumber(String registratrionNumber) {
    this.registratrionNumber = registratrionNumber;
  }

  public double getPrice() {
    return price;
  }
```

```
  public void setPrice(double price) {
    this.price = price;
  }

  public int getNoOfSeats() {
    return noOfSeats;
  }

  public void setNoOfSeats(int noOfSeats) {
    this.noOfSeats = noOfSeats;
  }

  @Override
  public int hashCode() {
    return registratrionNumber.hashCode();
  }

  @Override
  public boolean equals(Object o) {
    if(o == null)  //null can never be same as calling object so return false
      return false;
    if(this == o)   //here calling object and parameter object are same so return true
      return true;
    if(this.getClass() != o.getClass())  //check if both objects are of same class
      return false;
    //you are here means calling object and parameter object are of same class
    //they are pointing to different memory location, so match then on behalf of registrationNumber
    //Using reference 'o' of class Object, fields of class Car can't be accessed so we have to downcast 'o' to Car
    Car paraObj = (Car)o;
    return paraObj.registratrionNumber.equals(this.registratrionNumber);
  }

  @Override
  public String toString() {
    return "Registratrion Number: " + registratrionNumber + " Price: " + price + " Number of Seats: " + noOfSeats;
  }
}

package com.masai.sprint2;
class Tester {
  public static void main(String[] args) {
    Car carOne = new Car("MH-14 CM 5869", 700000.0, 7);
    Car carTwo = new Car("MH-14 CM 5869", 750000.0, 6);
    Car carThree = new Car("MH-14 JM 9657", 650000.0, 5);
    Car carFour = carOne;

    System.out.println(carOne == carTwo);  //false
```

```
    System.out.println(carTwo == carThree);  //false
    System.out.println(carThree == carOne);  //false
    System.out.println(carFour == carOne);  //true
    System.out.println(carTwo == carFour);  //false

    System.out.println();

    System.out.println("carOne.hashCode() = " + carOne.hashCode());
    System.out.println("carTwo.hashCode() = " + carTwo.hashCode());
    System.out.println("carThree.hashCode() = " + carThree.hashCode());
    System.out.println("carFour.hashCode() = " + carFour.hashCode());

    System.out.println();

    System.out.println("carOne.equals(carTwo) = " + carOne.equals(carTwo));
    System.out.println("carTwo.equals(carThree) = " + carTwo.equals(carThree));
    System.out.println("carThree.equals(carFour)          =          "          +
carThree.equals(carFour));
    System.out.println("carOne.equals(carFour) = " + carOne.equals(carFour));
    System.out.println("carTwo.equals(carFour) = " + carTwo.equals(carFour));

    System.out.println();

    System.out.println("For carOne:: " + carOne);
    System.out.println("For carTwo:: " + carTwo);
    System.out.println("For carThree:: " + carThree);
  }
}
```

Output

```
false
false
false
true
false

carOne.hashCode() = 1295006993
carTwo.hashCode() = 1295006993
carThree.hashCode() = -1082284623
carFour.hashCode() = 1295006993

carOne.equals(carTwo) = true
carTwo.equals(carThree) = false
carThree.equals(carFour) = false
carOne.equals(carFour) = true
carTwo.equals(carFour) = true

For carOne:: Registratrion Number: MH-14 CM 5869 Price: 700000.0 Number of
Seats: 7
For carTwo:: Registratrion Number: MH-14 CM 5869 Price: 750000.0 Number of
Seats: 6
For carThree:: Registratrion Number: MH-14 JM 9657 Price: 650000.0 Number of
Seats: 5
```

# You Activity

Create a class EPFOAccount that has private data members like accountNo and balance. Create parameterized constructor and override toString() method to return the textual presentation of the EPFOAccount object like
Account number: EPF001 Balance: 25500/-

Create a class Employee with private data members like employeeId, name, salary and epfoAccount (use has-A relationship). Override the equals(), hashcode() & toString() method such that
1. If two employee objects have same employeeId then equals() method should return true for them and hashcode of these objects must be same
2. The textual presentation of the Employee object be like
Employee Id: E001 Name: Anuj Salary: 45000 EPFO Account details: (Account number: EPF001 Balance: 25500/-)

```
class EPFOAccount {
  //write code here
}

class Employee{
  //write code here
}

class Main{
  static public void main(String args[]){
    EPFOAccount e1 = new EPFOAccount("EPFO001", 25000);
    EPFOAccount e2 = new EPFOAccount("EPFO002", 35000);

    Employee emp1 = new Employee ("E001", "Anuj", 45000, e1);
    Employee emp2 = new Employee ("E002", "Jayesh", 38000, e2);
    Employee emp3 = new Employee ("E001", "Rajesh", 35500, e2);

    System.out.println(emp1.equals(emp2));
    System.out.println(emp2.equals(emp3));
    System.out.println(emp3.equals(emp1));

    System.out.println();

    System.out.println(emp1.hashCode() == emp2.hashCode());
    System.out.println(emp2.hashCode() == emp3.hashCode());
    System.out.println(emp3.hashCode() == emp1.hashCode());

    System.out.println();

    System.out.println(emp1);
    System.out.println(emp2);
    System.out.println(emp3);
  }
}
```

Output

```
false
false
```

```
true

false
false
true

Employee Id: E001 Name: Anuj Salary: 45000 EPFO Account details: (Account number: EPFO001 Balance:  25000/-)

Employee Id: E002 Name: Jayesh Salary: 38000 EPFO Account details: (Account number: EPFO002 Balance:  35000/-)

Employee Id: E001 Name: Rajesh Salary: 35500 EPFO Account details: (Account number: EPFO002 Balance:  35000/-)
```