Week 2

01 July 2023 03:16 PM

01/07/2023

1) **Return type Object and typecasting**

```
public int m1(){
      return 10;
}
public Object m3(){
      return "abc";
}
public Object m2(){
      return 300;
}
```

```
A a1= new A();
int x= (int)a1.m2();
String y= (String) a1.m3();
```

Object class is the superclass. So it can return its child class's objects.

---

2) **super, this**
   - non-static keywords
   - cannot be used inside static; cannot be used inside main method
   - **Used for- i) Constructor calling ii) Variable calling iii) Method calling**

super-
   - **In constructor calling, it is used to call parent class's constructor.**

Every constructor's first line has by default super keyword. (If this is already present in constructor, then super is not present. For constructor calling, super and this don't exist together)

i) super and this in Constructor calling

```
public class A{
 public A(){
          System.out.println("Constructor-A");
      }
}
```

```
class B extends A{
      public B(int i){
      //by default super() is present
      System.out.println("Constructor-B-int");
      }
}
```

```
B b= new B();
```

Output:-
Constructor-A
Constructor-B

When do we need to write super() explicitly?
-> When parent class has only parameterized constructors, child class needs to write super(parameter).

★ THIS PROGRAM WILL GIVE ERROR-

Here, class A doesn't have default constructor. It only has parameterized constructor.
class B has default constructor given by java. That constructor will have super() which will try to call default constructor of class A; but as A doesn't have it, code will give error.

ERROR-

```
public class A{
      public A(int i){
            System.out.println("Constructor-A-int")
      }
}
```

```
public class B extends A{

}
```

```
public class B extends A{
	public B(){
		super(500); // now A's constructor will be called
		System.out.println("Constructor-B");
	}
}
```

this in constructor calling- used to call constructors of same class.
If n=no. of constructors in a class, then this can be used only in n-1 constructors.

```
class A{
	public A(){
		System.out.println("Constructor-A");
	}
}
```

Output:-
Constructor-A
Constructor-B-int
Constructor-B

```
class B extends A{
	public B(){
	// super and this don't come together in constructor calling
		this(5);
		System.out.println("Constructor-B");
	}
	public B(int i){
		// by default super() will be present
		System.out.prinln("Constructor-B-int");
	}

}
```

ii) super and this in Variable calling

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

class B extends A{
	int x= 30;

	public void printX(){
		int x= 40;
		System.out.println(x);// 40
		System.out.println(this.x); //30
		System.out.println(super.x); //10
	}
}
```

when "this" and "super" are used in main method-
Error- cannot be referenced from static context

iii) super and this in Method calling

```
public class A{
	public void m1(){
		System.out.println("m1-A");
	}
}

class B extends A{
	public void m1(){
		super.m1(); // parent class's method
		System.out.println("m1-B");
	}
	public void m2(){
```

```
        this.m1();
        super.m1(); //parent class's m1()
    }
}
```

---

3) Overriding

Syntax rules-

- Same return type, same method name, same parameters
- If return type is class type, then it should be same or its subclass
- Access modifier should be same or greater in child class.
  public > protected > default > private
- @Override- to check if the method can be overridden
- static, final and private methods cannot be overridden.

```
class A{
    protected void m2(){
    }
}

class B extends A{
    public void m2(){   //Access modifier should be same or greater in child class.
    }
}
```

---

i) private method in inheritance

NOT OVERRIDING

```
class A{
    private void m1(){  // as this is private method, it's only available to class A; so it can't be overridden.
    }
}

class B extends A{
    public void m1(){  // @Override will give error. Because this method is not overridden.
    }
}
```

---

ii) static method in inheritance

NOT OVERRIDING

```
class A{
        public static void m1(){  // belongs to class A only
    }
}

class B extends A{
    public static void m1(){  // @Override will give error. Belongs to class B only
    }

    //ERROR
    //public void m1(){}
}
```

---

iii) final method in inheritance

NOT OVERRIDING


```
class A{
        public final void m1(){
    }
}


class B extends A{
    // ERROR-
    //public final void m1(){
    //}
    //ERROR-
    //public void m1(){
    //}
}
```

Overloading in inherited class-

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

class B extends A{
    public void m1(int i){
    }
}
```

---

toString method in Object class returns hash code.
We can override toString() in our class.
It's purpose is to provide information about the object.

```
public class Student{
    int rollno;
    String name;

    public Student(int rn, String n){
        this.rollno= rn;
        this.name= n;
    }

    public String toString(){
        return "Roll No- "+rollno+ " Name- "+ name;
    }
}

class Test{
    public static void main(String[] args){
        Student s1= new Student();
        System.out.println(s1);
    }
}
```

---

main method can be overloaded. It cannot be overridden, because it is static.