PROGRAMMING

Lecture 12

Sushil Paudel

REVISE

When it’s been 7 hours and you still can’t understand your own code

PREVIOUS SEMSTER

- Class, Objects & Methods
- Conditional Statements & Loops
- Scanner, Constructor & Variables
- Encapsulation
- Inheritance
- Array & Array List

TODAY'S TOPIC

- Polymorphism
- Type of polymorphism
- Type Casting

POLYMORSHISM

- **Polymorphism in Java** is a concept by which we can perform a *single action in different ways*.
- Polymorphism is derived from 2 Greek words: poly and morphs.
- The word "poly" means many and "morphs" means forms.
- So polymorphism means many forms.

REAL TIME EXAMPLE

- A person at the same time can have different characteristic. Like a man at the same time can be a father, a son, a student, etc.
- So the same person posses different behavior in different situations. This is called polymorphism.
- Polymorphism is considered as one of the important features of Object Oriented Programming. Polymorphism allows us to perform a single action in different ways.

TYPES OF POLYMORHISM

In Java polymorphism is mainly divided into two types:

- Compile time Polymorphism
- Runtime Polymorphism

COMPILE TIME POLYMORHISM

- It is also known as static polymorphism. This type of polymorphism is achieved by method overloading or operator overloading.
- **Method Overloading**: When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**.
- Method can be overloaded by **change in number of arguments** or/and **change in type of arguments**.

METHOD OVERLOADING

- Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.
- It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.
- Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as addTwo(int,int) for two parameters, and

addThree(int a,int b,int c) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

WAYS OF METHOD OVERLOADING

1. No of parameters

```
public class Addition {
    public void add(int a, int b) {
    }

    public void add(int a, int b, int c) {
    }

    public void add(int a, int b, int c, int d) {
    }
}
```

WAYS OF METHOD OVERLOADING

2. Data type of parameters

```
public class Operations {
    public void add(double a, int b) {
    }

    public void add(double a, double b) {
    }

    public void add(String a, int b, int c) {
    }
}
```

EXAMPLE

```
public class Addition{
    public void add(int a, int b){
        System.out.println("Sum is: "+(a+b));
    }


    public void add(int a, int b, int c){
        System.out.println("Sum is: "+(a+b+c));
    }


    public void test(){
        add(1,2);
        add(1,2,3);
    }
}
```

OUTPUT

Sum is: 3

Sum is: 6

OPERATOR OVERLOADING

- Java also provide option to overload operators.
- For example, we can make the operator (‘+’) for string class to concatenate two strings.
- We know that this is the addition operator whose task is to add two operands.
- So a single operator ‘+’ when placed between integer operands, adds them and when placed between string operands, concatenates them.
- In java, Only “+” operator can be overloaded:
  - To add integers
  - To concatenate strings

EXAMPLE

```
class OperatorOverloading {

    void operator(String str1, String str2) {
        String s = str1 + str2;
        System.out.println("Concatenated String: " + s);
    }

    public void operator(int a, int b) {
        int c = a + b;
        System.out.println("Sum = " + c);
    }
}
```

RUNTIME POLYMORPHISM

- It is also known as Dynamic Method Dispatch.
- It is a process in which a function call to the overridden method is resolved at Runtime.
- This type of polymorphism is achieved by Method Overriding.
- Method overriding, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class.
- That base function is said to be overridden.

METHOD OVERRIDING

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

EXAMPLE (WITHOUT OVERRIDING

```
public class Vehicle{
        void run(){
                System.out.println("Vehicle is running");
        }
}
```

```
public class Bike extends Vehicle {
        public static void main(String args[]) {
                // creating an instance of child class
                Bike obj = new Bike();
                // calling the method with child class instance
                obj.run();
        }
}
```

Output

Vehicle is running

EXAMPLE (WITH OVERRIDING)

```
public class Vehicle{
        void run(){
                System.out.println("Vehicle is running");
        }
}
```

```
public class Bike extends Vehicle {
        // defining the same method as in the parent class
        void run() {
                System.out.println("Bike is running");
        }

        public static void main(String args[]) {
                Bike obj = new Bike();// creating object
                obj.run();// calling method
        }
}
```

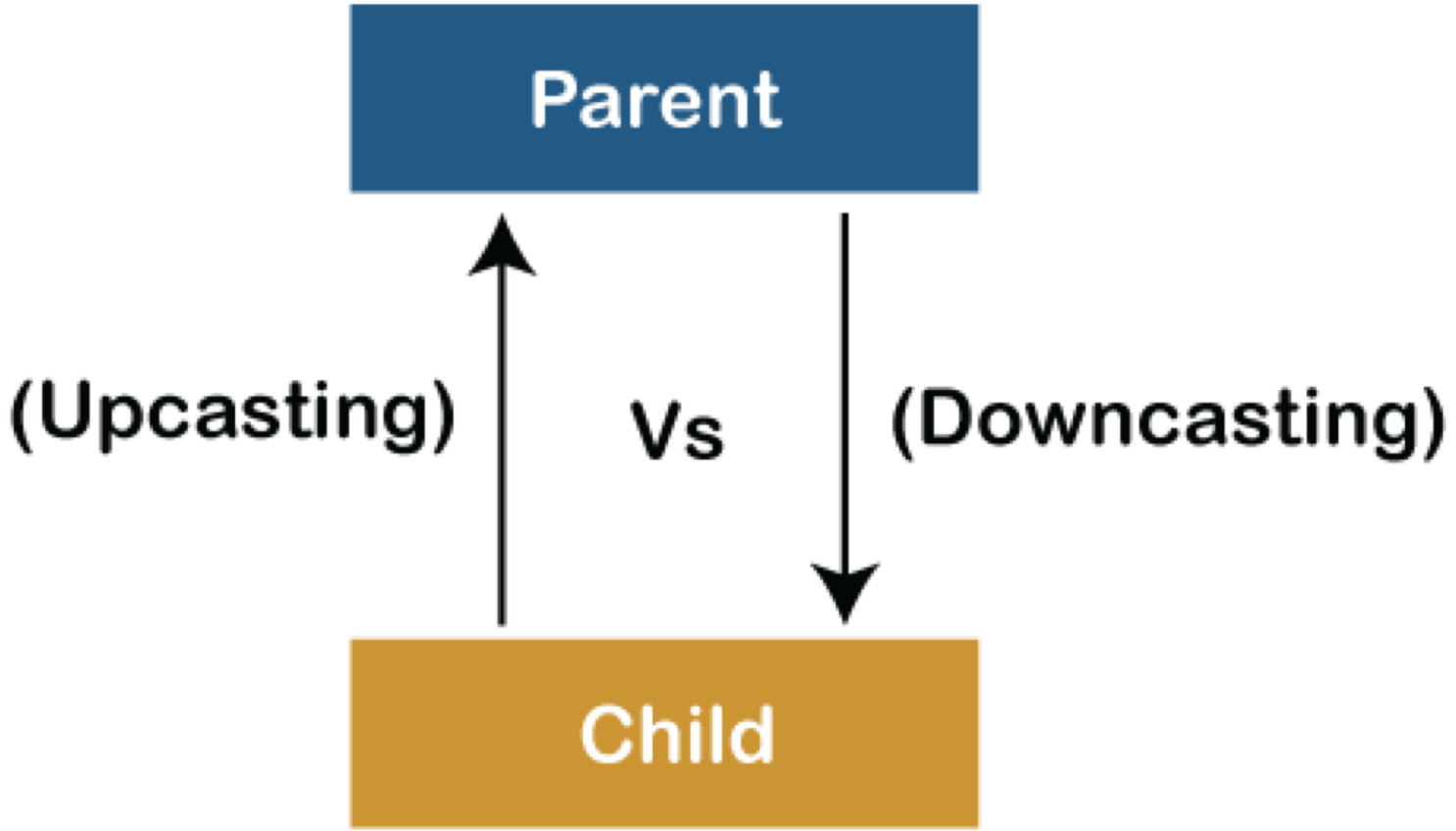
Output

Bike is running

CASTING

- Upcasting and Downcasting are purely related to inheritance hierarchy.
- If we are casting a subclass to a superclass type it is called ***upcasting*** and when a superclass type is casted to a subclass type it is called ***downcasting***.

CASTING



UPCASTING

- **Upcasting** is **casting** a **subtype** to a **supertype**, **upward** to the **inheritance tree**.
- Upcasting happens automatically and we don't have to explicitly do anything. Let us see a program snippet –

EXAMPLE

```
public class Parent{
    public void show() {
        System.out.println("Parent's show()");
    }
}
```

```
public class Child extends Parent{
    public void show() {
        System.out.println("Child's show()");
    }
}
```

```
public class Main{
    public static void main(String[] args) {
        Parent obj1 = new Child();
        obj1.show();
    }
}
```

OUTPUT

Child’s show()

DOWNCASTING

- When Subclass type refers to the object of Parent class, it is known as **downcasting**.
- If we perform it directly, compiler gives Compilation error.
- If you perform it by typecasting, ClassCastException is thrown at runtime.
- But if we use instanceof operator, downcasting is possible

EXAMPLE

```
class Animal {
}
```

```
class Dog3 extends Animal {
      public void method(Animal a) {
          if(a instanceof Dog3){
             Dog3 d = (Dog3) a;//downcasting
             System.out.println("ok downcasting performed");
          }
     }

     public static void main (String [] args) {
          Animal puppy = new Dog3();
          a.method(puppy);
     }
}
```

DOWNCASTING

```
Animal cat1 = new Cat();
Cat cat = (Cat) cat1;
```

Here, we cast the Animal type to the Cat type. As Cat is subclass of Animal, this casting is called downcasting. Unlike upcasting, downcasting can fail if the actual object type is not the target object type. For example:

```
Animal cat1 = new Cat();
Dog dog = (Dog) cat1;
```

This will throw a ClassCastException because the actual object type is Cat. And a Cat is not a Dog so we cannot cast it to a Dog.

DOWNCASTING

```
if (animal instanceof Cat) {
     Cat cat = (Cat) animal;
     cat.meow();
} else if (animal instanceof Dog) {
     Dog dog = (Dog) animal;
     dog.bark();
}
```

INSTANCE OF

- **instanceof returns false for null**

```
Test object = null;


if (object instanceof Test) {
	System.out.println(”object is instance of Test");
}else
	System.out.println(”object is NOT instance of Test");

}
```

Output

object is NOT instance of Test

INSTANCE OF

- A parent object is not an instance of Child

```
Parent object = new Parent();


if (object instanceof Child) {
        System.out.println("object is instance of Child");
}else{
        System.out.println("object is NOT instance of Child");
}
```

Output

obj is NOT instance of Child

INSTANCE OF

- **A parent reference referring to a Child is an instance of Child**

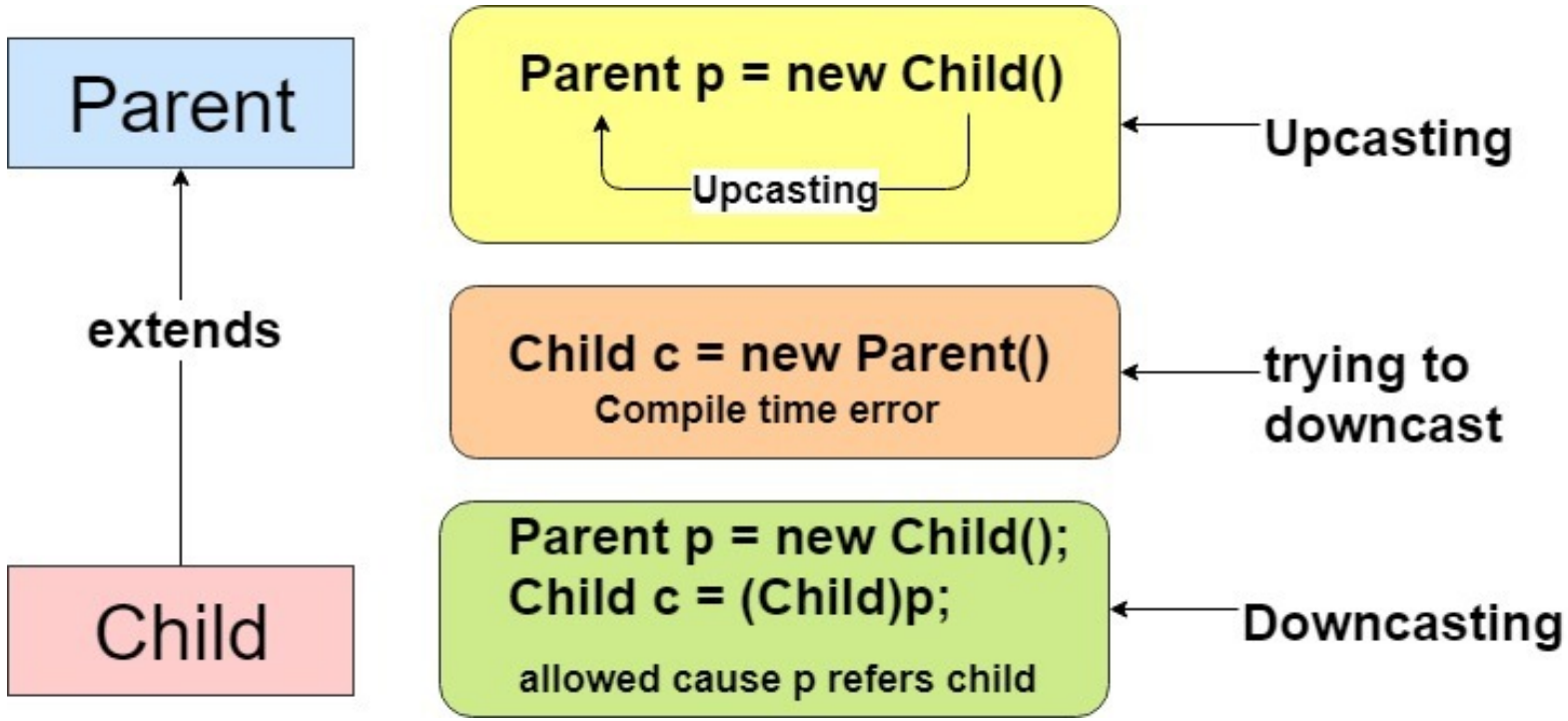
```
Parent par = new Child();

if (par instanceof Child){
        System.out.println(”Is instance”);
}
                }
```

Output

Is instance

DOWNCASTING



THANK YOU!

Any questions?