National University of Computer & Emerging Sciences,
Karachi
Computer Science Department
Spring 2022, Lab Manual – 06

Course Code: CL-1004	Course : Object Oriented Programming Lab
Instructor(s) :	Abeer Gauher, Hajra Ahmed, Syed Zain ul Hassan

LAB - 6

Type Casting and Inheritance

CASTING

Type Casting is a feature in Java using which the form or type of a variable or object is cast into some other kind or Object, and the process of conversion from one type to another is called Type Casting.

In Java, there are two types of casting:

Widening Casting (automatically) - converting a smaller type to a larger type size

byte -> short -> char -> int -> long -> float -> double

Narrowing Casting (manually) - converting a larger type to a smaller size type

double -> float -> long -> int -> char -> short -> byte

Implicit Type Casting

In **Implicit Type Casting** also known as **Widening Type Casting**, Java automatically converts one data type to another data type.

Example:

```
public class implicitcasting {
    public static void main(String [] args){
     // create int type variable
    int num = 10;
    System.out.println("The integer value: " + num);

    // convert into double type
    double data = num;
    System.out.println("The double value: " + data);
  }

}
```

Output:

```
The integer value: 10
The double value: 10.0
```

Java first converts the int type data into the double type. And then assign it to the double variable. The lower data type (having smaller size) is converted into the higher data type (having larger size). Hence there is no loss in data. This is why this type of conversion happens automatically.

Explicit Type Casting

In **Explicit Type Casting** also known as **Narrowing Type Casting**, we manually convert one data type into another using the parentheses.

Example:

```
public class explicitcasting {
    public static void main(String[] args) {
    // create double type variable
    double num = 10.99;
    System.out.println("The double value: " + num);

    // convert into int type
    int data = (int)num;
    System.out.println("The integer value: " + data);
  }

}
```

Output:

```
The double value: 10.99
The integer value: 10
```

In the above example, we are assigning the double type variable named num to an int type variable named data.

Notice the line,

int data = (int)num;

Here, the int keyword inside the parentheses indicates that that the num variable is converted into the int type.

In the case of Narrowing Type Casting, the higher data types (having larger size) are converted into lower data types (having smaller size). Hence there is the loss of data. This is why this type of conversion does not happen automatically.

Example: Type conversion from int to String

```
public class explicitcasting {
public static void main(String[] args) {
    // create int type variable
    int num = 10;
    System.out.println("The integer value is: " + num);

    // converts int to string type
    String data = String.valueOf(num);
    System.out.println("The string value is: " + data);
  }
}
```

Output:

```
The integer value is: 10
The string value is: 10
```

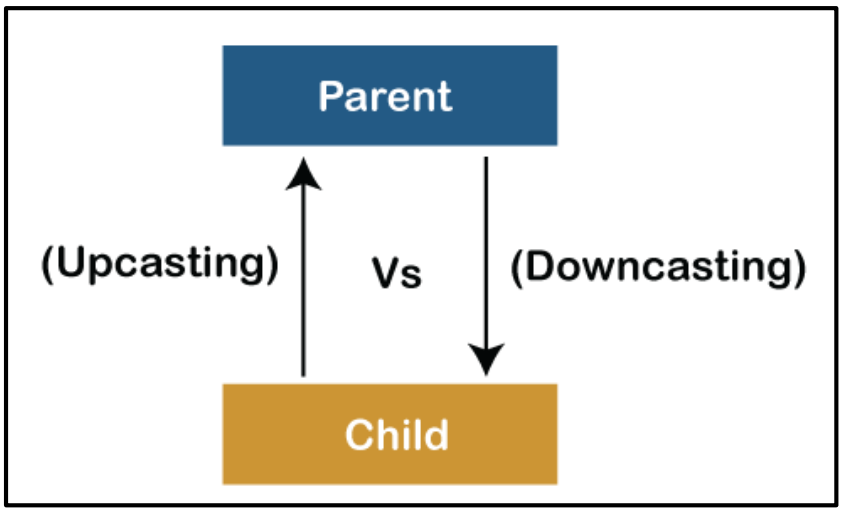
In the above program, notice the line

String data = String.valueOf(num);

Here, we have used the valueOf() method of the Java String class to convert the int type variable into a string.

Object Casting:

In Java, the object can also be typecasted like the datatypes. Parent and Child objects are two types of objects. So, there are two types of typecasting possible for an object, i.e., Child to Parent and Parent to Child or can say Upcasting and Downcasting.



Upcasting

Upcasting is a type of object typecasting in which a child object is typecasted to a parent class object. By using the Upcasting, we can easily access the variables and methods of the parent class to the child class. Here, we don't access all the variables and the method. We access only some specified variables and methods of the child class. Upcasting is also known as Generalization and Widening.

Example:

```
class  Parent{
   void PrintData() {
      System.out.println("method of parent class");
   }
}

class Child extends Parent {
   void PrintData() {
      System.out.println("method of child class");
   }
}

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

      Parent obj1 = (Parent) new Child();
      Parent obj2 = (Parent) new Child();
      obj1.PrintData();
      obj2.PrintData();
   }
}
```

Output:

```
method of child class
method of child class
```

Downcasting

In Java, we cannot assign a parent class reference object to the child class, but if we perform downcasting, we will not get any compile-time error. However, when we run it, it throws the "ClassCastException". Now the point is if downcasting is not possible in Java, then why is it allowed by the compiler? In Java, some scenarios allow us to perform downcasting. Here, the subclass object is referred by the parent class.

```
package com.mycompany.ooplab1;
class Parent {
    String name;

    // A method which prints the data of the parent class
    void showMessage()
    {
        System.out.println("Parent method is called");
    }
}

// Child class
class Child extends Parent {
    int age;

    // Performing overriding
    @Override
    void showMessage()
    {
        System.out.println("Child method is called");
    }
}
```

```
public class downcasting {
     public static void main(String[] args)
    {
        Parent p = new Child();
        p.name = "Java Lab";

        // Performing Downcasting Implicitly
        //Child c = new Parent(); // it gives compile-time error

        // Performing Downcasting Explicitly
        Child c = (Child)p;

        c.age = 10;
        System.out.println(c.name);
        System.out.println(c.age);
        c.showMessage();
    }

}
```

Inheritance:

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.

The idea behind inheritance in Java is that you can create new classesthat are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Terms used in Inheritance

- Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- Reusability: A mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class.

The syntax of Java Inheritance

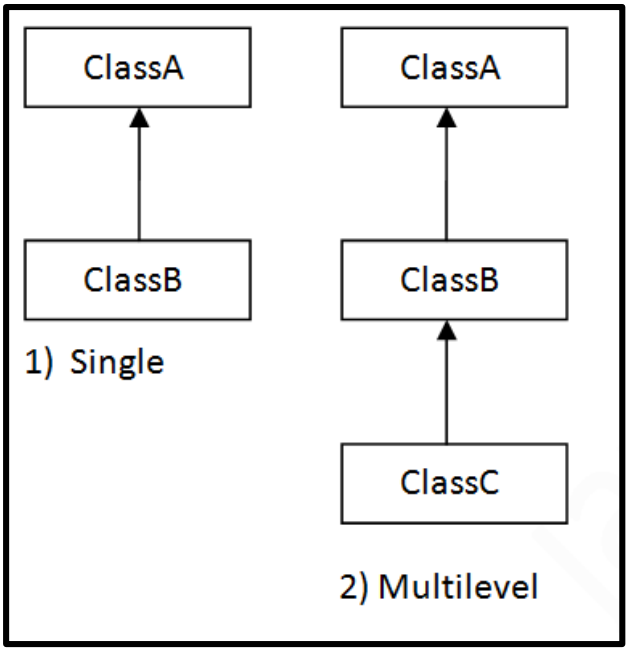
```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The extends keyword indicates that you are making a new class that derives from an existing class.

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only.



Single Inheritance Example

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

Using super to invoke constructors:

To explicitly call the superclass constructor from the subclass constructor, we use super(). It's a special form of the super keyword.

super() can be used only inside the subclass constructor and must be the first statement.

The compiler can automatically call the no-arg constructor. However, it cannot call parameterized constructors.

If a parameterized constructor has to be called, we need to explicitly define it in the subclass constructor.

```java
class Animal {
  // default or no-arg constructor
  Animal() {
    System.out.println("I am an animal in default constructor");
  }
  // parameterized constructor
  Animal(String type) {
    System.out.println("Type: "+type);
  }
}

class Dog extends Animal {
  // default constructor
  Dog() {
    //calling default constructor
    //super();
    // calling parameterized constructor of the superclass
    super("Animal");

    System.out.println("I am a dog");
  }
}
class TestInheritance {
  public static void main(String[] args) {
    Dog dog1 = new Dog();
  }
}
```

Output:

```
Type: Animal
I am a dog
```

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

```java
package com.mycompany.ooplab1;
class Data
{
int length,breadth,height,area,volume;
public void input(int l, int b, int ht)
    {
        length = l;
        breadth = b;
        height = ht;
   }
}
class Area extends Data
{
    public int calculateArea()
    {
        area = length*breadth;
        return area;
    }
}
class Volume extends Area
{
    public int calculateVolume()
    {
        volume = area*height;
        return volume;
    }
}
```

```java
public class MultilevelInheritance {
    public static void main(String args[]){
        Volume v = new Volume();
        v.input(5,15,2);

    int ans = v.calculateArea();
      System.out.println("Area of rectangle = "+ans);
      System.out.println("Volume of rectangle = "+v.calculateVolume());
}}
```

Output:

```
Area of rectangle = 75
Volume of rectangle = 150
```

LAB#06 EXERCISES

QUESTION#1

Write a Java program that has a class named “Course”.

- The class Course has the attributes course name, course code, class venue and credit hours, all are protected members.
- Set all these attributes with a parameterized constructor.
- Derive a class “Java Course” that has an attribute teacher name.
- Make a constructor and invoke the base class’s parameterized constructor.
- Set the teacher name in the constructor.
- The derived class has a function Display that displays all the details of the course and the derived class.
- In the main, display all the details.

QUESTION#2

Write a Java program that has a class named “Person”.

- The class has a default constructor that displays “I am a person”.
- The class has attributes name, age, nationality, address and CNIC.
- The class has an input function that prompts the user to enter all the details. For CNIC, the total number of digits should be exactly 13. If it’s less than 13 or greater display an error message.
- The class also has a display function that displays all the details.

Derive a class Employee from Person.

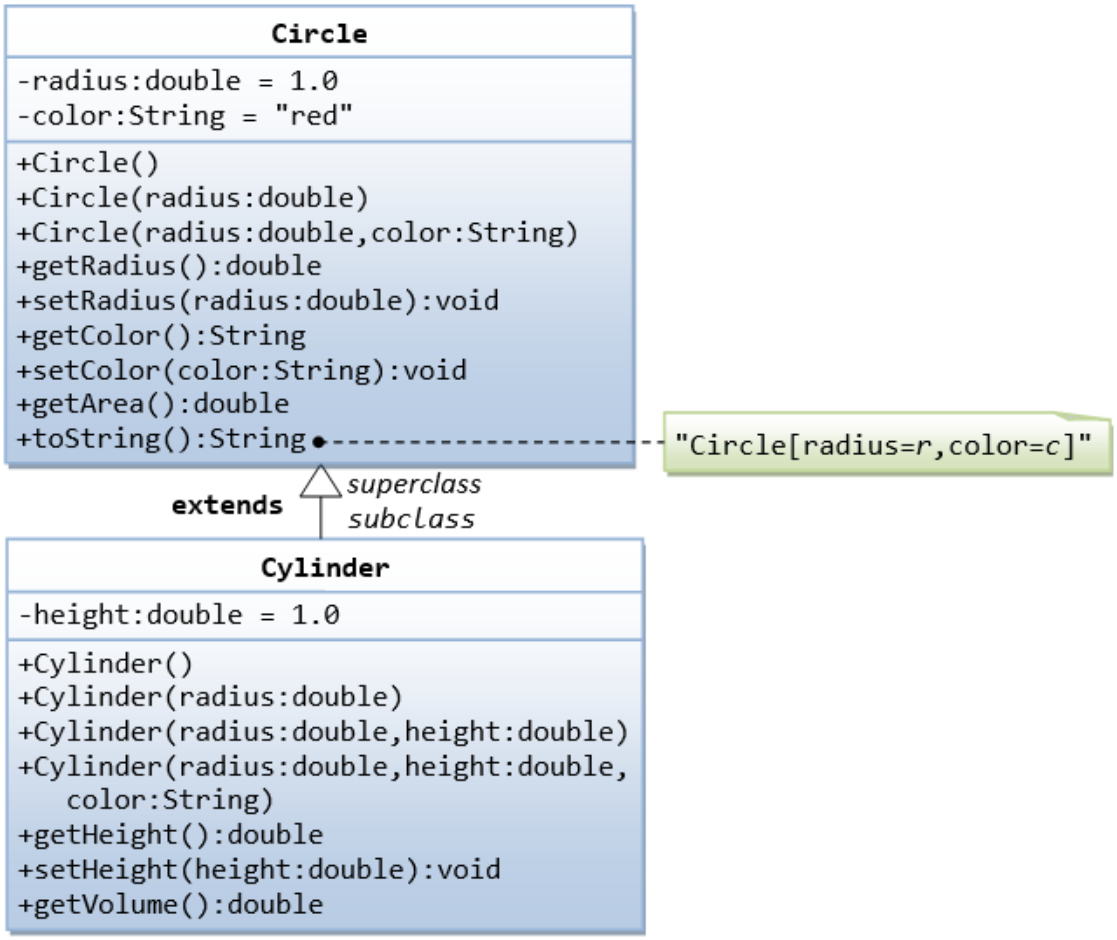
- The class Employee has a default constructor that invokes the base class’s constructor and displays “I am an Employee”.
- The class has the attributes name of company, company’s location (city), no of years worked.
- The class has an input function that prompts the user to enter all the details. It also has a display function that displays all the details.

Derive a class Manager from Employee.

- The class Manager has a default constructor that invokes the base class’s constructor and displays “I am a Manager”.
- The class has an array that contains the names of employee’s who are working under the manager’s supervision. Input atleast five employee’s in the array from the user and display all these employee’s too.

In the main program, call all the functions and display the details.

QUESTION#3



Implement the scenario given above in the class diagram.
Display the volume and area in double. Use typecasting and display the volume and area as an integer too.

QUESTION#4

- A library wants to organize its system by categorizing books such as Java, C, C++, etc. Implement a program that contains a base class called Books that will contain members such as book ID, book name, book author, ISBN and price. All are protected members.
- Derive one class from the base class and name it as “Category1”.
- The class has one data member that is the category.
- Make a parameterized constructor and invoke the base class’s constructor.
- Create a display function and display all the details of the books in Category1 (3 books).
- In the main program, perform object up casting and cast child object to a parent class object (Book).

QUESTION#5

Implement the scenario given in the figure.