# **Overriding toString() in Java**

// file name: Main.java

class Complex {

    private double re, im;

    public Complex(double re, double im) {

        this.re = re;

        this.im = im;

    }

}

// Driver class to test the Complex class

public class Main {

    public static void main(String[] args) {

        Complex c1 = new Complex(10, 15);

        System.out.println(c1);

    }

}

Output:

Complex@19821f

The output is, class name, then ‘at’ sign, and at the end [*hashCode*](http://en.wikipedia.org/wiki/Java_hashCode()) of object. All classes in Java inherit from the Object class, directly or indirectly (See point 1 of [this](https://www.geeksforgeeks.org/comparison-of-inheritance-in-c-and-java/)). The Object class has some basic methods like clone(), toString(), equals(),.. etc. The default toString() method in Object prints “class name @ hash code”. We can override toString() method in our class to print proper output. For example, in the following code toString() is overridden to print “Real + i Imag” form.

|  |
| --- |
| class Complex {      private double re, im;        public Complex(double re, double im) {          this.re = re;          this.im = im;      }        /\* Returns the string representation of this Complex number.         The format of string is "Re + iIm" where Re is real part         and Im is imagenary part.\*/      @Override      public String toString() {          return String.format(re + " + I " + im);      }  }    // Driver class to test the Complex class  public class Main {      public static void main(String[] args) {          Complex c1 = new Complex(10, 15);          System.out.println(c1);      }  } |

Output:

10.0 + I 15.0

<https://www.geeksforgeeks.org/overriding-tostring-method-in-java/>

<https://www.geeksforgeeks.org/overriding-equals-method-in-java/>

What is the purpose of toString() method in Java?

When you print an object, by default the Java compiler invokes the toString() method on the object. So by overriding the toString() method, we can provide meaningful output.

<https://devqa.io/overriding-tostring-java-class/>

Enums

An enum is a special "class" that represents a group of **constants** (unchangeable variables, like final variables).

To create an enum, use the enum keyword (instead of class or interface), and separate the constants with a comma. Note that they should be in uppercase letters:

### **Example**

enum Level {

LOW,

MEDIUM,

HIGH

}

You can access enum constants with the **dot** syntax:

Level myVar = Level.MEDIUM;

System.out.println(myVar);

OutPut is: MEDIUM

## Abstract Classes and Methods

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.  
Abstraction can be achieved with either **abstract classes** or [**interfaces**](https://www.w3schools.com/java/java_interface.asp) (which you will learn more about in the next chapter).

The abstract keyword is a non-access modifier, used for classes and methods:

* **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

* **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

abstract class Animal {

public abstract void animalSound();

public void sleep() {

System.out.println("Zzz");

}

}

From the example above, it is not possible to create an object of the Animal class:

Animal myObj = new Animal(); // will generate an error

### **Example**

// Abstract class

abstract class Animal {

// Abstract method (does not have a body)

public abstract void animalSound();

// Regular method

public void sleep() {

System.out.println("Zzz");

}

}

// Subclass (inherit from Animal)

class Cat extends Animal {

public void animalSound() {

// The body of animalSound() is provided here

System.out.println("The Cat says: wee wee");

}

}

class MyMainClass {

public static void main(String[] args) {

Cat myCat = new Cat(); // Create a Cat object

myCat.animalSound();

myCat.sleep();

}

}

**Result:**

The Cat says: wee wee  
Zzz

# **Super Keyword in Java**

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

1. **class** Animal{
2. String color="white";
3. }
4. **class** Cat **extends** Animal{
5. String color="black";
6. **void** printColor(){
7. System.out.println(color);//prints color of Cat class
8. System.out.println(**super**.color);//prints color of Animal class
9. }
10. }
11. **class** TestSuper1{
12. **public** **static** **void** main(String args[]){
13. Cat d=**new** Cat();
14. d.printColor();
15. }}

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** eat(){System.out.println("eating bread...");}
6. **void** bark(){System.out.println("barking...");}
7. **void** work(){
8. **super**.eat();
9. bark();
10. }
11. }
12. **class** TestSuper2{
13. **public** **static** **void** main(String args[]){
14. Dog d=**new** Dog();
15. d.work();
16. }}

Output:

eating...

barking...

3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

1. **class** Animal{
2. Animal(){System.out.println("animal is created");}
3. }
4. **class** Dog **extends** Animal{
5. Dog(){
6. **super**();
7. System.out.println("dog is created");
8. }
9. }
10. **class** TestSuper3{
11. **public** **static** **void** main(String args[]){
12. Dog d=**new** Dog();
13. }}
14. Output:
15. animal is created
16. dog is created

super example: real use

Let's see the real use of super keyword. Here, Emp class inherits Person class so all the properties of Person will be inherited to Emp by default. To initialize all the property, we are using parent class constructor from child class. In such way, we are reusing the parent class constructor.

1. **class** Person{
2. **int** id;
3. String name;
4. Person(**int** id,String name){
5. **this**.id=id;
6. **this**.name=name;
7. }
8. }
9. **class** Emp **extends** Person{
10. **float** salary;
11. Emp(**int** id,String name,**float** salary){
12. **super**(id,name);//reusing parent constructor
13. **this**.salary=salary;
14. }
15. **void** display(){System.out.println(id+" "+name+" "+salary);}
16. }
17. **class** TestSuper5{
18. **public** **static** **void** main(String[] args){
19. Emp e1=**new** Emp(1,"ankit",45000);
20. e1.display();
21. }}

**Output:**

1 ankit 45000

**HashSet:** If you don’t want to maintain insertion order but want store unique objects.  
**LinkedHashSet:** If you want to maintain insertion order of elements then you can use LinkedHashSet.

Java String format() Method Example 3

Apart from formatting, we can set width, padding etc. of any value. Let us see an example where we are setting width and padding for an integer value.

1. **public** **class** FormatExample3 {
2. **public** **static** **void** main(String[] args) {
3. String str1 = String.format("%d", 101);
4. String str2 = String.format("|%10d|", 101);  // Specifying length of integer
5. String str3 = String.format("|%-10d|", 101); // Left-justifying within the specified width
6. String str4 = String.format("|% d|", 101);
7. String str5 = String.format("|%010d|", 101); // Filling with zeroes
8. System.out.println(str1);
9. System.out.println(str2);
10. System.out.println(str3);
11. System.out.println(str4);
12. System.out.println(str5);
13. }
14. }

**OutPut:**

101

| 101|

|101 |

| 101|

|0000000101|

%s is used to convert any type of data into string

<https://www.javatpoint.com/java-string-format>