Object Oriented Programming

By Watcharin Sarachai

Topic

- Abstract Class
- Interface
- instanceof Keyword

Abstract Class

Abstract Class

- A class that is declared using “abstract” keyword is known as **abstract class**.

- It can have abstract methods (methods without body) as well as concrete methods (regular methods with body).

- A normal class (non-abstract class) cannot have abstract methods.

An abstract class can not be instantiated, which means you are not allowed to create an object of it.

Why we need an abstract class?

- Lets say we have a class Animal that has a method sound() and the subclasses (see inheritance) of it like Dog, Lion, Horse, Cat etc.

- Since the animal sound differs from one animal to another, there is no point to implement this method in parent class.

- This is because every child class must override this method to give its own implementation details, like Lion class will say “Roar” in this method and Dog class will say “Woof”.

Why we need an abstract class?

- When we know that all the animal child classes will and should override this method, then there is no point to implement this method in parent class.

- Thus, making this method abstract would be the good choice as by making this method abstract.

- This force all the sub classes to implement this method (*otherwise you will get compilation error*)

- We don’t need to give any implementation to this method in parent class.

Abstraction

Abstraction hides complexity from the user and shows only the relevant information. The details of an abstract method are implemented separately by each class.

Goals

- Code Reusability
- Flexibility of Implementation
- Multiple Inheritance

In Java

Abstract Classes (Partial Abstraction) and Interfaces (Total Abstraction)

Syntax:

Abstract classes: `abstract` | Interfaces: `interface` `implements`

Abstract class Example

- Now each animal must have a sound, by making this method abstract.
- We made it compulsory to the child class to give implementation details to this method. This way we ensures that every animal has a sound.

```java
//abstract parent class
abstract class Animal {
  // abstract method
  public abstract void sound();
}

// Dog class extends Animal class
public class Dog extends Animal {

  public void sound() {
    System.out.println("Woof");
  }

  public static void main(String args[]) {
    Animal obj = new Dog();
    obj.sound();
  }
}
```

```
Output:

Woof
```

Hence for such kind of scenarios we generally declare the class as abstract and later concrete classes extend these classes and override the methods accordingly and can have their own methods as well.

Abstract class declaration

- An abstract class outlines the methods but not necessarily implements all the methods.

```
//Declaration using abstract keyword
abstract class A {
  // This is abstract method
  abstract void myMethod();

  // This is concrete method with body
  void anotherMethod() {
    // Does something
  }
}
```

Abstract class declaration

```
abstract class Animal {
  // abstract methods
  abstract void move();
  abstract void eat();

  // concrete method
  void label() {
    System.out.println("Animal's data:");
  }
}
class Bird extends Animal {

  void move() {
    System.out.println("Moves by flying.");
  }

  void eat() {
    System.out.println("Eats birdfood.");
  }
}
```

Abstract class declaration

```java
abstract class Animal {
  // abstract methods
  abstract void move();
  abstract void eat();

  // concrete method
  void label() {
    System.out.println("Animal's data:");
  }
}
class Bird extends Animal {

  void move() {
    System.out.println("Moves by flying.");
  }

  void eat() {
    System.out.println("Eats birdfood.");
  }
}
```

```java
class Fish extends Animal {
  void move() {
    System.out.println("Moves by swimming.");
  }

  void eat() {
    System.out.println("Eats seafood.");
  }
}
```

Abstract class declaration

```
class TestBird {
  public static void main(String[] args) {
    Animal myBird = new Bird();

    myBird.label();
    myBird.move();
    myBird.eat();
  }
}

class TestFish {
  public static void main(String[] args) {
    Animal myFish = new Fish();

    myFish.label();
    myFish.move();
    myFish.eat();
  }
}
```

Abstract Class Rules

- **Note 1**: As we seen in the previous example, there are cases when it is difficult or often unnecessary to implement all the methods in parent class.

- In these cases, we can declare the parent class as abstract, which makes it a special class which is not complete on its own.

- A class derived from the abstract class must implement all those methods that are declared as **abstract** in the **parent class**.

Abstract Class Rules

- **Note 2**: Abstract class cannot be instantiated which means you cannot create the object of it.

    - To use this class, you need to create another class that extends this this class and provides the implementation of abstract methods.

    - Then you can use the object of that child class to call non-abstract methods of parent class as well as implemented methods (*those that were abstract in parent but implemented in child class*).

- **Note 3**: If a child does not implement all the abstract methods of abstract parent class, then the child class must need to be declared abstract as well.

Why can’t we create the object of an abstract class?

- Because these classes are incomplete, they have abstract methods that have no body.

- So if java allows you to create object of this class then if someone calls the abstract method using that object then What would happen? There would be no actual implementation of the method to invoke.

- Also because an object is **concrete**. An abstract class is like a **template**, so we have to extend it and build on it before you can use it.

Example to demonstrate that object creation of abstract class is not allowed

- As discussed above, we cannot instantiate an abstract class. This program throws a compilation error.

```
abstract class AbstractDemo {
  public void myMethod() {
    System.out.println("Hello");
  }

  abstract public void anotherMethod();
}

public class Demo extends AbstractDemo {

  public void anotherMethod() {
    System.out.print("Abstract method");
  }

  public static void main(String args[]) {
    // error: You can't create object of it
    AbstractDemo obj = new AbstractDemo();
    obj.anotherMethod();
  }
}
```

```
Output:

Unresolved compilation problem:
Cannot instantiate the type
AbstractDemo
```

Note: The class that extends the abstract class, have to implement all the abstract methods of it, else you have to declare that class abstract as well.

Abstract class vs Concrete class

- A class which is not abstract is referred as Concrete class.

- In the previous example that we have seen in the beginning of this course, **Animal** is a abstract class and **Cat**, **Dog** & **Lion** are concrete classes.

- **Key Points:**

1. An abstract class has no use until unless it is extended by some other class.

2. If we declare an abstract method in a class then you must declare the class abstract as well. you can't have abstract method in a concrete class. It's vice versa is not always true: ***If a class is not having any abstract method then also it can be marked as abstract***.

3. It can have non-abstract method (concrete) as well.

Abstract class vs Concrete class

- Abstract method has no body.
- Always end the declaration with a semicolon (;).
- It must be overridden. An abstract class must be extended and in a same way **abstract method must be overridden**.
- A class has to be declared abstract to have abstract methods.

Note: The class which is extending abstract class must override all the abstract methods.

Example of Abstract class and method

```
abstract class MyClass{
  public void disp(){
    System.out.println("Concrete method of parent class");
  }
  abstract public void disp2();
}

class Demo extends MyClass{
  /* Must Override this method while extending
   * MyClas
   */
  public void disp2()
  {
      System.out.println("overriding abstract method");
  }
  public static void main(String args[]){
      Demo obj = new Demo();
      obj.disp2();
  }
}
```

```
Output:

overriding abstract method
```

Interface

Interface

- As we discussed abstract class in the previous slide which is used for achieving partial abstraction. Unlike abstract class an interface is used for **full abstraction**.

- Abstraction is a process where you show only "**relevant**" data and "**hide**" unnecessary details of an object from the user.

- Interface looks like a class but it is not a class.

- An interface can have methods and variables just like the class but **the methods declared in interface are by default abstract**.

- The variables declared in an interface are **public, static & final by default**.

What is the use of interface in Java?

- As mentioned in previous slide they are used for full abstraction.

- Since methods in interfaces do not have body, they have to be implemented by the class before you can access them.

- The class that implements interface must implement all the methods of that interface.

- Also, java programming language does not allow you to extend more than one class, **However you can implement more than one interfaces in your class**.

Interface Syntax:

- Interfaces are declared by specifying a keyword “interface”:

```
interface MyInterface {
  /*
   * All the methods are public abstract by default As you see they have no body
   */
  public void method1();

  public void method2();
}
```

Example of an Interface in Java

```java
interface MyInterface {
  /*
   * compiler will treat them as: public abstract void method1(); public abstract
   * void method2();
   */
  public void method1();

  public void method2();
}

class Demo implements MyInterface {
  /*
   * This class must have to implement both the abstract methods else you will get
   * compilation error
   */
  public void method1() {
    System.out.println("implementation of method1");
  }

  public void method2() {
    System.out.println("implementation of method2");
  }

  public static void main(String arg[]) {
    MyInterface obj = new Demo();
    obj.method1();
  }
}
```

```
Output:

implementation of method1
```

Example of an Interface in Java

```
public interface Animal {
  public void eat();
  public void sound();
}
public interface Bird {
  int numberOfLegs = 2;
  String outerCovering = "feather";

  public void fly();
}
public class Eagle implements Animal, Bird {
  public void eat() {
    System.out.println("Eats reptiles and amphibians.");
  }
  public void sound() {
    System.out.println("Has a high-pitched whistling sound.");
  }
  public void fly() {
    System.out.println("Flies up to 10,000 feet.");
  }
}
```

Example of an Interface in Java

```
public class TestEagle {
  public static void main(String[] args) {
    Eagle myEagle = new Eagle();

    myEagle.eat();
    myEagle.sound();
    myEagle.fly();

    System.out.println("Number of legs: " + Bird.numberOfLegs);
    System.out.println("Outer covering: " + Bird.outerCovering);
  }
}
```

Interface and Inheritance

- An interface can not implement another interface. It has to **extend** the other interface.

- See the below example where we have two interfaces **Inf1** and **Inf2**. **Inf2** extends **Inf1** so If class implements the **Inf2** it has to provide implementation of all the methods of interfaces **Inf2** as well as **Inf1**.

Interface and Inheritance

```java
interface Inf1 {
  public void method1();
}

interface Inf2 extends Inf1 {
  public void method2();
}

public class Demo2 implements Inf2 {
  /*
   * Even though this class is only implementing the interface Inf2, it has to
   * implement all the methods of Inf1 as well because the interface Inf2 extends
   * Inf1
   */
  public void method1() {
    System.out.println("method1");
  }

  public void method2() {
    System.out.println("method2");
  }

  public static void main(String args[]) {
    Inf2 obj = new Demo();
    obj.method2();
  }
}
```

In this program, the class Demo only implements interface Inf2, however it has to provide the implementation of all the methods of interface Inf1 as well, because interface Inf2 extends Inf1.

Tag or Marker interface in Java

- An empty interface is known as **tag** or **marker** interface.
- For example **Serializable**, **EventListener**, **Remote** (java.rmi.Remote) are tag interfaces.
- These interfaces do not have any field and methods in it.

Interfaces Key Points:

1. We can’t instantiate an interface in java. That means we cannot create the object of an interface

2. Interface provides full abstraction as none of its methods have body. On the other hand abstract class provides partial abstraction as it can have abstract and concrete (methods with body) methods both.

3. implements keyword is used by classes to implement an interface.

4. While providing implementation in class of any method of an interface, it needs to be mentioned as public.

5. Class that implements any interface must implement all the methods of that interface, else the class should be declared abstract.

6. Interface cannot be declared as private, protected or transient.

7. All the interface methods are by default abstract and public.

Interfaces Key Points:

8.Variables declared in interface are public, static and final by default.

```
interface Try {
  int a = 10;
  public int a = 10;
  public static final int a = 10;
  final int a = 10;
  static int a = 0;
}
```

9.Interface variables must be initialized at the time of declaration otherwise compiler will throw an error.

```
interface Try {
  int x;// Compile-time error
}
```

Interfaces Key Points:

8.Variables declared in interface are public, static and final by default.

```
interface Try {
  int a = 10;
  public int a = 10;
  public static final int a = 10;
  final int a = 10;
  static int a = 0;
}
```

9.Interface variables must be initialized at the time of declaration otherwise compiler will throw an error.

```
interface Try {
  int x;// Compile-time error
}
```

Above code will throw a compile time error as the value of the variable x is not initialized at the time of declaration.

Interfaces Key Points:

10. Inside any implementation class, you cannot change the variables declared in interface because by default, they are public, static and final.

```
class Sample implements Try {
  public static void main(String args[]) {
    x = 20; // compile time error
  }
}
```

11. An interface can extend any interface but cannot implement it. Class implements interface and interface extends interface.

12. A class can implement any number of interfaces.

Interfaces Key Points:

13. If there are two or more same methods in two interfaces and a class implements both interfaces, implementation of the method once is enough.

```
interface A {
  public void aaa();
}

interface B {
  public void aaa();
}

class Central implements A, B {
  public void aaa() {
    // Any Code here
  }

  public static void main(String args[]) {
    // Statements
  }
}
```

Interfaces Key Points:

13. If there are two or more same methods in two interfaces and a class implements both interfaces, implementation of the method once is enough.

```
interface A {
  public void aaa();
}

interface B {
  public void aaa();
}

class Central implements A, B {
  public void aaa() {
    // Any Code here
  }

  public static void main(String args[]) {
    // Statements
  }
}
```

Interfaces Key Points:

14. A class cannot implement two interfaces that have methods with same name but different return type.

```
interface A {
  public void aaa();
}

interface B {
  public int aaa();
}

class Central implements A, B {
  public void aaa() // error
  {
  }
  public int aaa() // error
  {
  }
  public static void main(String args[]) {

  }
}
```

Interfaces Key Points:

15. Variable names conflicts can be resolved by interface name.

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

interface B {
  int x = 100;
}

class Hello implements A, B {
  public static void Main(String args[]) {
    /*
     * reference to x is ambiguous both variables are x so we are using interface
     * name to resolve the variable
     */
    System.out.println(x);
    System.out.println(A.x);
    System.out.println(B.x);
  }
}
```

Difference Between Abstract Class and Interface in Java

	Abstract Class	Interface
1	An abstract class can extend only one class or one abstract class at a time	An interface can extend any number of interfaces at a time
2	An abstract class can extend another concrete (regular) class or abstract class	An interface can only extend another interface
3	An abstract class can have both abstract and concrete methods	An interface can have only abstract methods
4	In abstract class keyword “abstract” is mandatory to declare a method as an abstract	In an interface keyword “abstract” is optional to declare a method as an abstract
5	An abstract class can have protected and public abstract methods	An interface can have only have public abstract methods
6	An abstract class can have static, final or static final variable with any **access specifier**	interface can only have public static final (constant) variable

Abstract class vs interface in Java

- Difference No.1: Abstract class can extend only one class or one abstract class at a time

```
class Example1 {
  public void display1() {
    System.out.println("display1 method");
  }
}

abstract class Example2 {
  public void display2() {
    System.out.println("display2 method");
  }
}

abstract class Example3 extends Example1 {
  abstract void display3();
}
```

```
class Example4 extends Example3 {
  public void display3() {
    System.out.println("display3 method");
  }
}

class Demo {
  public static void main(String args[]) {
    Example4 obj = new Example4();
    obj.display3();
  }
}
```

```
Output:

display3 method
```

Abstract class vs interface in Java

- Difference No.1: Abstract class can extend only one class or one abstract class at a time

```
//first interface
interface Example1 {
  public void display1();
}

// second interface
interface Example2 {
  public void display2();
}
```

```
// This interface is extending both the above interfaces
interface Example3 extends Example1, Example2 {
}

class Example4 implements Example3 {
  public void display1() {
    System.out.println("display2 method");
  }

  public void display2() {
    System.out.println("display3 method");
  }
}

class Demo {
  public static void main(String args[]) {
    Example4 obj = new Example4();
    obj.display1();
  }
}
```

```
Output:

display2 method
```

Interface can extend any number of interfaces at a time

Abstract class vs interface in Java

- Difference No.2: Abstract class can be extended (inherited) by a class or an abstract class

```
class Example1 {
  public void display1() {
    System.out.println("display1 method");
  }
}

abstract class Example2 {
  public void display2() {
    System.out.println("display2 method");
  }
}

abstract class Example3 extends Example2 {
  abstract void display3();
}
```

```
class Example4 extends Example3 {
  public void display2() {
    System.out.println("Example4-display2 method");
  }

  public void display3() {
    System.out.println("display3 method");
  }
}

class Demo {
  public static void main(String args[]) {
    Example4 obj = new Example4();
    obj.display2();
  }
}
```

```
Output:

Example4-display2 method
```

Abstract class vs interface in Java

- Difference No.2: Abstract class can be extended (inherited) by a class or an abstract class

```
interface Example1 {
  public void display1();
}

interface Example2 extends Example1 {
}

class Example3 implements Example2 {
  public void display1() {
    System.out.println("display1 method");
  }
}

class Demo {
  public static void main(String args[]) {
    Example3 obj = new Example3();
    obj.display1();
  }
}
```

```
Output:

display1 method
```

Interfaces can be extended only by interfaces. Classes has to implement them instead of extend

Abstract class vs interface in Java

- Difference No.3: Abstract class can have both abstract and concrete methods

```
abstract class Example1 {
  abstract void display1();

  public void display2() {
    System.out.println("display2 method");
  }
}

class Example2 extends Example1 {
  public void display1() {
    System.out.println("display1 method");
  }
}

class Demo {
  public static void main(String args[]) {
    Example2 obj = new Example2();
    obj.display1();
  }
}
```

Abstract class vs interface in Java

- Difference No.3: Abstract class can have both abstract and concrete methods

```
interface Example1 {
  public abstract void display1();
}

class Example2 implements Example1 {
  public void display1() {
    System.out.println("display1 method");
  }
}

class Demo {
  public static void main(String args[]) {
    Example2 obj = new Example2();
    obj.display1();
  }
}
```

```
Output:

display1 method
```

Interface can only have abstract methods, they cannot have concrete methods

Abstract class vs interface in Java

- Difference No.4: In abstract class, the keyword ‘abstract’ is mandatory to declare a method as an abstract

```
abstract class Example1 {
  public abstract void display1();
}

class Example2 extends Example1 {
  public void display1() {
    System.out.println("display1 method");
  }

  public void display2() {
    System.out.println("display2 method");
  }
}

class Demo {
  public static void main(String args[]) {
    Example2 obj = new Example2();
    obj.display1();
  }
}
```

Abstract class vs interface in Java

- Difference No.4: In abstract class, the keyword ‘abstract’ is mandatory to declare a method as an abstract

```
interface Example1 {
  public void display1();
}

class Example2 implements Example1 {
  public void display1() {
    System.out.println("display1 method");
  }

  public void display2() {
    System.out.println("display2 method");
  }
}

class Demo {
  public static void main(String args[]) {
    Example2 obj = new Example2();
    obj.display1();
  }
}
```

In interfaces, the keyword ‘abstract’ is optional to declare a method as an abstract because all the methods are abstract by default

Abstract class vs interface in Java

- Difference No.5: Abstract class can have protected and public abstract methods

```
abstract class Example1 {
  protected abstract void display1();

  public abstract void display2();

  public abstract void display3();
}
```

```
class Example2 extends Example1 {
  public void display1() {
    System.out.println("display1 method");
  }

  public void display2() {
    System.out.println("display2 method");
  }

  public void display3() {
    System.out.println("display3 method");
  }
}

class Demo {
  public static void main(String args[]) {
    Example2 obj = new Example2();
    obj.display1();
  }
}
```

Abstract class vs interface in Java

- Difference No.5: Abstract class can have protected and public abstract methods

```
interface Example1 {
  void display1();
}

class Example2 implements Example1 {
  public void display1() {
    System.out.println("display1 method");
  }

  public void display2() {
    System.out.println("display2 method");
  }
}

class Demo {
  public static void main(String args[]) {
    Example2 obj = new Example2();
    obj.display1();
  }
}
```

Interface can have only public abstract methods

Abstract class vs interface in Java

- Difference No.6: Abstract class can have static, final or static final variables with any access specifier

```
abstract class Example1 {
  private int numOne = 10;
  protected final int numTwo = 20;
  public static final int numThree = 500;

  public void display1() {
    System.out.println("Num1=" + numOne);
  }
}

class Example2 extends Example1 {
  public void display2() {
    System.out.println("Num2=" + numTwo);
    System.out.println("Num2=" + numThree);
  }
}

class Demo {
  public static void main(String args[]) {
    Example2 obj = new Example2();
    obj.display1();
    obj.display2();
  }
}
```

Abstract class vs interface in Java

- Difference No.6: Abstract class can have static, final or static final variables with any access specifier

```
interface Example1 {
  int numOne = 10;
}

class Example2 implements Example1 {
  public void display1() {
    System.out.println("Num1=" + numOne);
  }
}

class Demo {
  public static void main(String args[]) {
    Example2 obj = new Example2();
    obj.display1();
  }
}
```

Interface can have only public static final (constant) variable

instanceof Keyword

- The instanceof operator compares an object to a specified type.
    - You can use it to test if an object is:
        - an instance of a class,
        - an instance of a subclass,
        - or an instance of a class that implements a particular interface.

instanceof Example

- Here class Lion and Cow extends Animal

```
public class Animal {
    // Body hidden
}
```

```
public class Cow extends Animal {
    // Body hidden
}
```

```
public class Lion extends Animal {
// Body hidden
public void roar() { /* Body hidden */ }
}
```

```
public static void main(String[] args) {
  Animal[] zoo= new Animal[2];
  zoo[0] = new Cow();
  zoo[1] = new Lion();
  for( int i =0; i<zoo.length; i++) {
    Animal a = zoo[i];
    if( a instanceof Lion) { //test using instanceof keyword
      System.out.println("Animal " + i + " is a Lion");
      Lion l = (Lion) a; //Cast the Object to a Lion
      l.roar(); //Call a method in the Lion class
    }
  }
}
```

Lab 6

- From the Lab05-EXERCISE:
  - Methods that are the same in a subclass must be moved to one place in the superclass. So, the method that depends on subclass behavior must be marked as an abstract method.
  - The abstract method defined in the superclass must be implemented in the subclass according to its behavior.
  - The Circle and Rectangle classes can be resized, so we must implement them with a Resizable interface with the "resize()" method. The "resize(double factor)" take one parameter name "factor".
- Write a main method to:
  - Create and initialize a Circle, Rectangle, Triangle, or Square object and append it to an array list.
  - Print the area & perimeter of all object in the array list
  - If the user chooses to resize menu, all object in array that implements Resizable interface must resize by factor parameter that the user has given.
  - Re-print area & perimeter of all objects in the array list.

```
1.Circle
2.Rectangle
3.Triangle
4.Square
5.Resize
6.Exit
Please select [1-5]: 1
Enter the radius: 5

[The Circle area is : 78.54, and the perimeter is 31.42]
```

```
1.Circle
2.Rectangle
3.Triangle
4.Square
5.Resize
6.Exit
Please select [1-5]: 2
Enter the length: 4
Enter the breadth: 3

[The Circle area is : 78.54, and the perimeter is 31.42]
[The Rectangle area is : 12.00, and the perimeter is 14.00]
```

```
1.Circle
2.Rectangle
3.Triangle
4.Square
5.Resize
6.Exit
Please select [1-5]: 4
Enter the side: 6

[The Circle area is : 78.54, and the perimeter is 31.42]
[The Rectangle area is : 12.00, and the perimeter is 14.00]
[The Square area is : 36.00, and the perimeter is 24.00]
```

```
1.Circle
2.Rectangle
3.Triangle
4.Square
5.Resize
6.Exit
Please select [1-5]: 3
Enter the base: 2
Enter the height: 3

[The Circle area is : 78.54, and the perimeter is 31.42]
[The Rectangle area is : 12.00, and the perimeter is 14.00]
[The Square area is : 36.00, and the perimeter is 24.00]
[The Triangle area is : 3.00, and the perimeter is 3.61]
```

```
1.Circle
2.Rectangle
3.Triangle
4.Square
5.Resize
6.Exit
Please select [1-5]: 1
Enter the radius: 3

[The Circle area is : 78.54, and the perimeter is 31.42]
[The Rectangle area is : 12.00, and the perimeter is 14.00]
[The Square area is : 36.00, and the perimeter is 24.00]
[The Triangle area is : 3.00, and the perimeter is 3.61]
[The Circle area is : 28.27, and the perimeter is 18.85]
```

```
1.Circle
2.Rectangle
3.Triangle
4.Square
5.Resize
6.Exit
Please select [1-5]: 5
Please enter factor to resize: 0.5

[The Circle area is : 19.63, and the perimeter is 15.71]
[The Rectangle area is : 3.00, and the perimeter is 7.00]
[The Square area is : 9.00, and the perimeter is 12.00]
[The Triangle area is : 3.00, and the perimeter is 3.61]
[The Circle area is : 7.07, and the perimeter is 9.42]
```

Reference

- **Super keyword in java with examplehttps://beginnersbook.com/2014/07/super-keyword-in-java-with-example/**
  Accessed: 2020-07-07
- **6 OOP Concepts in Java with examples [2020] · Raygun Bloghttps://raygun.com/blog/oop-concepts-java/**
  Accessed: 2020-07-04

```
}
```

```
class JavaExample extends ParentClass {
  JavaExample() {
    /*
     * It by default invokes the constructor of parent class You can use super() to
     * call the constructor of parent. It should be the first statement in the child
     * class constructor, you can also call the parameterized constructor of parent
     * class by using super like this: super(10), now this will invoke the
     * parameterized constructor of int arg
     */
    System.out.println("Constructor of Child");
  }
```

```
  public static void main(String args[]) {
```

Output:

```
Constructor of Parent
Constructor of Child
```