**Java – Abstraction**

As per dictionary, **abstraction** is the quality of dealing with ideas rather than events. For example, when you consider the case of e-mail, complex details such as what happens as soon as you send an e-mail, the protocol your e-mail server uses are hidden from the user. Therefore, to send an e-mail you just need to type the content, mention the address of the receiver, and click send.

Likewise in Object-oriented programming, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

In Java, abstraction is achieved using Abstract classes and interfaces.

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Before learning java abstract class, let's understand the abstraction in java first.

**Abstraction in Java**

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

**Ways to achieve Abstaction**

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

**Abstract Class**

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

**abstract** **class** A{}

* Abstract classes may or may not contain *abstract methods*, i.e., methods without body ( public void get(); )
* But, if a class has at least one abstract method, then the class **must** be declared abstract.
* If a class is declared abstract, it cannot be instantiated.
* To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
* If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

**abstract method**

|  |
| --- |
| A method that is declared as abstract and does not have implementation is known as abstract method. |

**Example abstract method**

**abstract** **void** printStatus();//no body and abstract

**Example of abstract class that has abstract method**

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

**abstract** **class** Bike{

**abstract** **void** run();

}

***class****Honda4****extends****Bike{*

***void****run(){*

*System.out.println("running safely..");*

*}*

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

*Bike obj =****new****Honda4();*

*obj.run();*

*}*

*}*

**Output:**

running safely..

**Understanding the real scenario of abstract class**

In this example, Shape is the abstract class, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the **factory method**.

A **factory method** is the method that returns the instance of the class. We will learn about the factory method later.

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

*File: TestAbstraction1.java*

***abstract******class****Shape{*

***abstract******void****draw();*

*}*

*//In real scenario, implementation is provided by others i.e. unknown by end user*

***class****Rectangle****extends****Shape{*

***void****draw(){*

*System.out.println("drawing rectangle");*

*}*

*}*

***class****Circle1****extends****Shape{*

***void****draw(){*

*System.out.println("drawing circle");*

*}*

*}*

*//In real scenario, method is called by programmer or user*

***class****TestAbstraction1{*

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

*Shape s=****new****Circle1();*

*//In real scenario, object is provided through method e.g. getShape() method  s.draw();*

*}*

*}*

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestAbstraction1)

drawing circle

### Another example of abstract class in java

File: TestBank.java

*abstract int getRateOfInterest();*

*}*

*class SBI extends Bank{*

*abstract class Bank{*

*int getRateOfInterest(){*

*return 7;*

*}*

*}*

*class PNB extends Bank{*

*int getRateOfInterest(){*

*return 8;*

*}*

*}*

*class TestBank{*

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

*Bank b;*

*b=new SBI();*

*System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");*

*b=new PNB();*

*System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");*

*}*

*}*

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestBank)

Rate of Interest is: 7 %

Rate of Interest is: 8 %

### Abstract class having constructor, data member, methods etc.

An abstract class can have data member, abstract method, method body, constructor and even main() method.

File: TestAbstraction2.java

*//example of abstract class that have method body*

*abstract class Bike{*

*Bike(){*

*System.out.println("bike is created");}*

*abstract void run();*

*void changeGear(){*

*System.out.println("gear changed");*

*}*

*}*

*class Honda extends Bike{*

*void run(){*

*System.out.println("running safely..");*

*}*

*}*

*class TestAbstraction2{*

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

*Bike obj = new Honda();*

*obj.run();*

*obj.changeGear();*

*}*

*}*

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestAbstraction2)

bike is created

running safely..

gear changed

#### Rule: If there is any abstract method in a class, that class must be abstract.

*class Bike12{*

*abstract void run();*

*}*

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Bike12)

compile time error

#### Rule: If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

### Another real scenario of abstract class

The abstract class can also be used to provide some implementation of the interface. In such case, the end user may not be forced to override all the methods of the interface.

#### Note: If you are beginner to java, learn interface first and skip this example.

*interface A{*

*void a();*

*void b();*

*void c();*

*void d();*

*}*

*abstract class B implements A{*

*public void c(){*

*System.out.println("I am C");*

*}*

*}*

*class M extends B{*

*public void a(){*

*System.out.println("I am a");*

*}*

*public void b(){*

*System.out.println("I am b");*

*}*

*public void d(){*

*System.out.println("I am d");*

*}*

*}*

*class Test5{*

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

*A a=new M();*

*a.a();*

*a.b();*

*a.c();*

*a.d();*

*}*

*}*

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Test5)

**Output:** I am a

I am b

I am c

I am d

**Example**

This section provides you an example of the abstract class. To create an abstract class, just use the **abstract** keyword before the class keyword, in the class declaration.

**/\* File name : Employee.java \*/**

*public abstract class Employee {*

*private String name;*

*private String address;*

*private int number;*

*public Employee(String name, String address, int number) {*

*System.out.println("Constructing an Employee");*

*this.name = name;*

*this.address = address;*

*this.number = number;*

*}*

*public double computePay() {*

*System.out.println("Inside Employee computePay");*

*return 0.0;*

*}*

*public void mailCheck() {*

*System.out.println("Mailing a check to " + this.name + " " + this.address);*

*}*

*public String toString() {*

*return name + " " + address + " " + number;*

*}*

*public String getName() {*

*return name;*

*}*

*public String getAddress() {*

*return address;*

*}*

*public void setAddress(String newAddress) {*

*address = newAddress;*

*}*

*public int getNumber() {*

*return number;*

*}*

*}*

You can observe that except abstract methods the Employee class is same as normal class in Java. The class is now abstract, but it still has three fields, seven methods, and one constructor.

Now you can try to instantiate the Employee class in the following way −

**/\* File name : AbstractDemo.java \*/**

*public class AbstractDemo {*

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

***/\* Following is not allowed and would raise error \*/***

*Employee e = new Employee("George W.", "Houston, TX", 43);*

*System.out.println("\n Call mailCheck using Employee reference--");*

*e.mailCheck();*

*}*

*}*

When you compile the above class, it gives you the following error −

*Employee.java:46: Employee is abstract; cannot be instantiated*

*Employee e = new Employee("George W.", "Houston, TX", 43);*

*^*

*1 error*

**Inheriting the Abstract Class**

We can inherit the properties of Employee class just like concrete class in the following way −

**Example**

**/\* File name : Salary.java \*/**

*public class Salary extends Employee {*

*private double salary; // Annual salary*

*public Salary(String name, String address, int number, double salary) {*

*super(name, address, number);*

*setSalary(salary);*

*}*

*public void mailCheck() {*

*System.out.println("Within mailCheck of Salary class ");*

*System.out.println("Mailing check to " + getName() + " with salary " + salary);*

*}*

*public double getSalary() {*

*return salary;*

*}*

*public void setSalary(double newSalary) {*

*if(newSalary >= 0.0) {*

*salary = newSalary;*

*}*

*}*

*public double computePay() {*

*System.out.println("Computing salary pay for " + getName());*

*return salary/52;*

*}*

*}*

Here, you cannot instantiate the Employee class, but you can instantiate the Salary Class, and using this instance you can access all the three fields and seven methods of Employee class as shown below.

***/\* File name : AbstractDemo.java \*/***

*public class AbstractDemo {*

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

*Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);*

*Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);*

*System.out.println("Call mailCheck using Salary reference --");*

*s.mailCheck();*

*System.out.println("\n Call mailCheck using Employee reference--");*

*e.mailCheck();*

*}*

*}*

This produces the following result −

Output

Constructing an Employee

Constructing an Employee

Call mailCheck using Salary reference --

Within mailCheck of Salary class

Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--

Within mailCheck of Salary class

Mailing check to John Adams with salary 2400.0

**Abstract Methods**

If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as an abstract.

* **abstract** keyword is used to declare the method as abstract.
* You have to place the **abstract** keyword before the method name in the method declaration.
* An abstract method contains a method signature, but no method body.
* Instead of curly braces, an abstract method will have a semoi colon (;) at the end.

**Following is an example of the abstract method.**

**Example**

*public abstract class Employee {*

*private String name;*

*private String address;*

*private int number;*

*public abstract double computePay();*

*// Remainder of class definition*

*}*

**Declaring a method as abstract has two consequences −**

* The class containing it must be declared as abstract.
* Any class inheriting the current class must either override the abstract method or declare itself as abstract.

**Note** − Eventually, a descendant class has to implement the abstract method; otherwise, you would have a hierarchy of abstract classes that cannot be instantiated.

Suppose Salary class inherits the Employee class, then it should implement the **computePay()** method as shown below −

***/\* File name : Salary.java \*/***

*public class Salary extends Employee {*

*private double salary; // Annual salary*

*public double computePay() {*

*System.out.println("Computing salary pay for " + getName());*

*return salary/52;*

*}*

*// Remainder of class definition*

*}*

**What is Abstraction?**  
  
Abstraction is process of **hiding the implementation details** and showing only the functionality.  
  
Abstraction in java is achieved by using interface and abstract class. Interface give 100% abstraction and abstract class give 0-100% abstraction.  
  
**What is Abstract class in Java?**  
  
A class that is declared as **abstract** is known as abstract class.  
  
**Syntax:** abstract class <class-name>{}  
  
An abstract class is something which is incomplete and you cannot create instance of abstract class.  
If you want to use it you need to make it complete or concrete by extending it.

A class is called concrete if it does not contain any abstract method and implements all abstract method inherited from abstract class or interface it has implemented or extended.

**What is Abstract method in Java?**  
  
A method that is declare as abstract and **does not have implementation** is known as abstract method.  
 If you define abstract method than class must be abstract.  
  
Syntax:  
 abstract return\_type method\_name ();  
  
An abstract method in Java doesn't have body, it’s just a declaration. In order to use abstract method you need to **override** that method in Subclass.  
  
Example 1 :( Without abstract method)

class Employee extends Person {   
 private String empCode;   
 public String getEmpCode() {   
 return empCode;   
 }   
 public void setEmpCode(String empCode) {   
 this.empCode = empCode;   
 }   
}   
abstract class Person {   
 private String name;   
 public String getName() {   
 return name;   
 }   
 public void setName(String name) {   
 this.name = name;   
 }   
 }   
public class Main{   
 public static void main(String args[]){   
*//INSTIATING AN ABSTRACT CLASS GIVES COMPILE TIME ERROR   
//Person p = new Person() ;   
//THIS REFERENCE VARIABLE CAN ACESS ONLY THOSE METHOD WHICH ARE OVERRIDDEN*   
 Person person = new Employee();   
 person.setName("Jatin Kansagara");   
 System.out.println(person.getName());   
 }   
}

Example 2: (with abstract method)

public class Main{   
 public static void main(String args[]){   
 TwoWheeler test = new Honda();   
 test.run();   
 }   
 }   
 abstract class TwoWheeler {   
 public abstract void run();   
 }   
 class Honda extends TwoWheeler{   
 public void run(){   
 System.out.println("Running..");   
 }   
 }

**When do you use abstraction?**  
  
When you know something needs to be there but not sure how exactly it should look like.

**Advantages of Abstraction**  
  
By using abstraction, we can s**eparate the things** that can be grouped to another type.  
  
Frequently changing properties and methods can be grouped to a separate type so that the main type need not undergo changes. This adds strength to the OOAD principle -"**Code should be open for Extension but closed for Modification**".  
  
Simplifies the representation of the **domain models**.  
  
**Summary:**  
  
-    Use abstraction if you know something needs to be in class but implementation of that varies.  
-     In Java you **cannot create instance of abstract class** , its compiler error.  
-    abstract is a keyword in java.  
-    A class automatically becomes abstract class when any of its method declared as abstract.  
-     abstract method doesn't have method body.  
-    **Variable cannot be made abstract**, its only behavior or methods which would be abstract.  
-    If a class extends an abstract class or interface it has to provide implementation to all its abstract method to be a concrete class. Alternatively this class can also be abstract.

## Types of abstraction

Typically abstraction can be seen in two ways:

#### 1) Data abstraction

Data abstraction is the way to create complex data types and exposing only meaningful operations to interact with data type, where as hiding all the implementation details from outside works.

Benefit of this approach involves capability of improving the implementation over time e.g. solving performance issues is any. The idea is that such changes are not supposed to have any impact on client code, since they involve no difference in the abstract behavior.

#### 2) Control abstraction

A software is essentially a collection of numerous statements written in any programming language. Most of the times, statement are similar and repeated over places multiple times.

Control abstraction is the process of identifying all such statements and expose them as a unit of work. We normally use this feature when we create a function to perform any work.

## How to use abstraction in java

As abstraction is one of core principles of Object oriented programming practices, and Java following all OOPs principles, abstraction is one of major building block of java language.

**Data abstraction** spans from creating simple data objects to complex collection implementations such as [HashMap](http://howtodoinjava.com/core-java/collections/how-hashmap-works-in-java/) or HashSet. Similarly, **control abstraction** can be seen from defining simple function calls to complete open source frameworks. control abstraction is main force behind [structured programming](https://en.wikipedia.org/wiki/Structured_programming).

### Abstract class in Java Important Points

1. abstract keyword is used to create an abstract class in java.
2. Abstract class in java can’t be instantiated.
3. We can use abstract keyword to create an abstract method, an abstract method doesn’t have body.
4. If a class have abstract methods, then the class should also be abstract using abstract keyword, else it will not compile.
5. It’s not necessary to have abstract class to have abstract method.
6. If abstract class doesn’t have any method implementation, its better to use interface because java doesn’t support multiple class inheritance.
7. The subclass of abstract class in java must implement all the abstract methods unless the subclass is also an abstract class.
8. All the methods in an interface are implicitly abstract unless the interface methods are static or default. Static methods and default methods in interfaces are added in Java 8, for more details read [Java 8 interface changes](http://www.journaldev.com/2752/java-8-interface-changes-static-method-default-method).
9. Java Abstract class can implement interfaces without even providing the implementation of interface methods.
10. Java Abstract class is used to provide common method implementation to all the subclasses or to provide default implementation.
11. We can run abstract class in java like any other class if it has main() method.
12. Here is a simple example of Abstract Class in Java.
13. Notice that work() is an abstract method and it has no body. Here is a concrete class example extending abstract class in java.

package com.journaldev.design;  
public class Employee extends Person {

private int empId;

public Employee(String nm, String gen, int id) {

super(nm, gen);

this.empId=id;

}

@Override

public void work() {

if(empId == 0){

System.out.println("Not working");

}else{

System.out.println("Working as employee!!");

}

}

public static void main(String args[]){

//coding in terms of abstract classes

Person student = new Employee("Dove","Female",0);

Person employee = new Employee("Pankaj","Male",123);

student.work();

employee.work();

//using method implemented in abstract class - inheritance

employee.changeName("Pankaj Kumar");

System.out.println(employee.toString());

}

}

### Abstract class

If a class contain any abstract method then the class is declared as abstract class. An abstract class is never instantiated. It is used to provide abstraction. Although it does not provide 100% abstraction because it can also have concrete method.

**Syntax :**

**abstract** class *class\_name* { }

#### Abstract method

Method that are declared without any body within an abstract class are called **abstract method**. The method body will be defined by its subclass. Abstract method can never be final and static. Any class that extends an abstract class must implement all the abstract methods declared by the super class.

**Syntax :**

**abstract** return\_type *function\_name* (); // No definition

#### Example of Abstract class

***abstract*** *class A{*

***abstract*** *void callme();*

*}*

*class B extends A{*

*void callme() {*

*System.out.println("this is callme.");*

*}*

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

*B b = new B();*

*b.callme();*

*}*

*}*

**Output:**

this is callme.

#### Abstract class with concrete(normal) method.

Abstract classes can also have normal methods with definitions, along with abstract methods.

***abstract*** *class A{*

***abstract*** *void callme();*

*public void normal(){*

*System.out.println("this is concrete method");*

*}*

*}*

*class B extends A{*

*void callme() {*

*System.out.println("this is callme.");*

*}*

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

*B b = new B();*

*b.callme();*

*b.normal();*

*}*

*}*

**Output**

this is callme.

this is concrete method.

#### Points to Remember

1. Abstract classes are not Interfaces. They are different, we will study this when we will study Interfaces.
2. An abstract class may or may not have an abstract method. But if any class has even a single abstract method, then it must be declared abstract.
3. Abstract classes can have Constructors, Member variables and Normal methods.
4. Abstract classes are never instantiated.
5. When you extend Abstract class with abstract method, you must define the abstract method in the child class, or make the child class abstract.

#### Abstraction using abstract class

Abstraction is an important feature of OOPS. It means hiding complexity. Abstract class is used to provide abstraction. Although it does not provide 100% abstraction because it can also have concrete method.

Let’s see how abstract class is used to provide abstraction.

***abstract*** *class Vehicle{*

*public* ***abstract*** *void engine();*

*}*

*public class Car extends Vehicle {*

*public void engine(){*

*System.out.println("Car engine");*

*//car engine implementation*

*}*

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

*Vehicle v = new Car();*

*v.engine();*

*}*

*}*

**Output**

Car engine

Here by casting instance of **Car** type to **Vehicle** reference, we are hiding the complexity of **Car** type under **Vechicle**. Now the **Vehicle** reference can be used to provide the implementation but it will hide the actual implementation process.

#### When to use Abstract Methods & Abstract Class?

Abstract methods are usually declared where two or more subclasses are expected to do a similar thing in different ways through different implementations. These subclasses extend the same Abstract class and provide different implementations for the abstract methods.

Abstract classes are used to define generic types of behaviors at the top of an object-oriented programming class hierarchy, and use its subclasses to provide implementation details of the abstract class.

**Using an Abstract class**

The purpose of an abstract class is to specify the default functionality of an object and let its sub-classes to explicitly implement that functionality. Thus, it stands as an abstraction layer that must be extended and implemented by the corresponding sub-classes.

A sample example of using an abstract class is the following. We declare an abstract class, called Instrument:

*Instrument.java:*

|  |  |
| --- | --- |
|  | abstract class Instrument { |
|  | protected String name; |

|  |  |
| --- | --- |
| 3 |  |
| 4 | abstract public void play(); |

|  |  |
| --- | --- |
| 5 | } |

As we can observe, an Instrument object contains a field name and a method called play, that must be implemented by a sub-class.

Next, we define a sub-class called StringedInstrument that extends the Instrument class and adds an extra field called numberOfStrings:

*StringedInstrument.java:*

[view source](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#viewSource)[print](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#printSource)[?](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#about)

|  |  |
| --- | --- |
| 1 | abstract class StringedInstrument extends Instrument { |
| 2 | protected int numberOfStrings; |

|  |  |
| --- | --- |
| 3 | } |

Finally, we add two more classes that implement the functionality of a StringedIntrument, called ElectricGuitar and ElectricBassGuitar accordingly. The definition of these newly added classes is shown below:

*ElectricGuitar.java:*

[view source](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#viewSource)[print](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#printSource)[?](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#about)

|  |  |
| --- | --- |
| 01 | public class ElectricGuitar extends StringedInstrument { |
| 02 |  |

|  |  |
| --- | --- |
| 03 | public ElectricGuitar() { |
| 04 | super(); |

|  |  |
| --- | --- |
| 05 | this.name = "Guitar"; |
| 06 | this.numberOfStrings = 6; |

|  |  |
| --- | --- |
| 07 | } |
| 08 |  |

|  |  |
| --- | --- |
| 09 | public ElectricGuitar(int numberOfStrings) { |
| 10 | super(); |

|  |  |
| --- | --- |
| 11 | this.name = "Guitar"; |
| 12 | this.numberOfStrings = numberOfStrings; |

|  |  |
| --- | --- |
| 13 | } |
| 14 |  |

|  |  |
| --- | --- |
| 15 | @Override |
| 16 | public void play() { |

|  |  |
| --- | --- |
| 17 | System.out.println("An electric " + numberOfStrings + "-string " + name |
| 18 | + " is rocking!"); |

|  |  |
| --- | --- |
| 19 | } |
| 20 | } |

*ElectricBassGuitar.java:*

[view source](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#viewSource)[print](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#printSource)[?](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#about)

|  |  |
| --- | --- |
| 01 | public class ElectricBassGuitar extends StringedInstrument { |
| 02 |  |

|  |  |
| --- | --- |
| 03 | public ElectricBassGuitar() { |
| 04 | super(); |

|  |  |
| --- | --- |
| 05 | this.name = "Bass Guitar"; |
| 06 | this.numberOfStrings = 4; |

|  |  |
| --- | --- |
| 07 | } |
| 08 |  |

|  |  |
| --- | --- |
| 09 | public ElectricBassGuitar(int numberOfStrings) { |
| 10 | super(); |

|  |  |
| --- | --- |
| 11 | this.name = "Bass Guitar"; |
| 12 | this.numberOfStrings = numberOfStrings; |

|  |  |
| --- | --- |
| 13 | } |
| 14 |  |

|  |  |
| --- | --- |
| 15 | @Override |
| 16 | public void play() { |

|  |  |
| --- | --- |
| 17 | System.out.println("An electric " + numberOfStrings + "-string " + name |
| 18 | + " is rocking!"); |

|  |  |
| --- | --- |
| 19 | } |
| 20 | } |

Finally, we create a new class called Execution that contains a single main method:

*Execution.java:*

[view source](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#viewSource)[print](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#printSource)[?](https://examples.javacodegeeks.com/java-basics/java-abstract-class-example/#about)

|  |  |
| --- | --- |
| 01 | import main.java.music.ElectricBassGuitar; |
| 02 | import main.java.music.ElectricGuitar; |

|  |  |
| --- | --- |
| 03 |  |
| 04 | public class Execution { |

|  |  |
| --- | --- |
| 05 |  |
| 06 | public static void main(String[] args) { |

|  |  |
| --- | --- |
| 07 | ElectricGuitar guitar = new ElectricGuitar(); |
| 08 | ElectricBassGuitar bassGuitar = new ElectricBassGuitar(); |

|  |  |
| --- | --- |
| 09 |  |
| 10 | guitar.play(); |

|  |  |
| --- | --- |
| 11 | bassGuitar.play(); |
| 12 |  |

|  |  |
| --- | --- |
| 13 | guitar = new ElectricGuitar(7); |
| 14 | bassGuitar = new ElectricBassGuitar(5); |

|  |  |
| --- | --- |
| 15 |  |
| 16 | guitar.play(); |

|  |  |
| --- | --- |
| 17 | bassGuitar.play(); |
| 18 | } |

|  |  |
| --- | --- |
| 19 | } |

In this example, we create two different instances of an ElectricGuitar and an ElectricBassGuitar classes and we call their play methods. A sample execution of the aforementioned main method is shown below:

An electric 6-string Guitar is rocking!

An electric 4-string Bass Guitar is rocking!

An electric 7-string Guitar is rocking!

An electric 5-string Bass Guitar is rocking!

[Abstraction In Java](http://javaconceptoftheday.com/abstraction-in-java/)

In the computer science perspective, Abstraction is the process of separating ideas from their action.

Yes, In the computer science, Abstraction is used to separate ideas from their implementation. Abstraction in java is used to define only ideas in one class so that the idea can be implemented by its sub classes according to their requirements. For example,

[?](http://javaconceptoftheday.com/abstraction-in-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | abstract class Animal  {      abstract void soundOfAnimal();  // It is just an idea  }    class Cat extends Animal  {      void soundOfAnimal()      {          System.out.println("Meoh");          //Implementation of the idea according to requirements of sub class      }  }    class Dog extends Animal  {      void soundOfAnimal()      {          System.out.println("Bow Bow");          //Implementation of the idea according to requirements of sub class      }  } |

Abstraction in java is implemented using Abstract classes and interfaces.

Today we will discuss only Abstract Classes. In the next concept, we will discuss about interfaces.

**Abstract Classes :**

Abstract classes contain abstract methods (you can refer them as ideas) so that they can be implemented in sub classes according to their requirements. They are also called as incomplete classes as they have some unimplemented abstract methods(ideas).

Let’s discuss some rules need to follow while using abstract classes and abstract methods.

* Abstract classes and abstract methods are declared using ‘**abstract**‘ keyword. We can’t create objects to those classes which are declared as abstract. But, we can create objects to sub classes of abstract class, provided they must implement abstract methods.

[?](http://javaconceptoftheday.com/abstraction-in-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | abstract class AbstractClass  {      abstract void abstractMethod();  }    class ConcreteClass extends AbstractClass  {      void abstractMethod()      {          System.out.println(&quot;Abstract Method Implemented&quot;);      }  }    public class Abstraction  {      public static void main(String[] args)      {          //AbstractClass A = new AbstractClass();  Can't create objects to Abstract class          ConcreteClass C = new ConcreteClass();          //ConcreteClass implements abstract method,          //so we can create object to ConcreteClass          AbstractClass A1 = C;          //ConcreteClass object is auto-upcasted to AbsractClass      }  } |

* The methods which are not implemented or which don’t have definitions must be declared with ‘abstract’ keyword and the class which contains it must be also declared as abstract.

[?](http://javaconceptoftheday.com/abstraction-in-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | // It gives compile time error  class AbstractClass  {      void abstractMethod();  //This method must be declared as abstract or must be defined      abstract void abstractMethod();  //The Class must be also declared as abstract  }    //      \*\*\*\*\*   \*\*\*\*\*     \*\*\*\*\*   \*\*\*\*\*  // This is OK  abstract class AbstractClass  {      abstract void abstractMethod();  } |

* It is not compulsory that abstract class must have abstract methods. It may or may not have abstract methods. But the class which has at least one abstract method must be declared as abstract.

[?](http://javaconceptoftheday.com/abstraction-in-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | abstract class AbstractClass  {      void methodOne()      {          //Concrete Method      }      //No Abstract methods but class is abstract  } |

* You can’t create objects to abstract class even though it does not contain any abstract methods.

[?](http://javaconceptoftheday.com/abstraction-in-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | abstract class AbstractClass  {      void methodOne()      {          //Concrete Method      }        void methodTwo()      {          //Concrete Method      }  }    public class Abstraction  {      public static void main(String[] args)      {          AbstractClass a = new AbstractClass();  //Compile time error          //You can't create objects to AbstractClass          //even though it does not contain any abstract methods.      }  } |

* Abstract Class can be a combination of concrete and abstract methods.

[?](http://javaconceptoftheday.com/abstraction-in-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | abstract class AbstractClass  {      void methodOne()      {          //Concrete Method      }        void methodTwo()      {          //Concrete Method      }        abstract void methodThree();  //Abstract Method        abstract void methodFour();  //Abstract Method  } |

* Any class extending an abstract class must implement all abstract methods. If it does not implement, it must be declared as abstract.

[?](http://javaconceptoftheday.com/abstraction-in-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | abstract class AbstractClass  {      abstract void abstractMethodOne();  //Abstract Method        abstract void abstractMethodTwo();  //Abstract Method  }    class ConcreteClass extends AbstractClass  {      void abstractMethodOne()      {          //abstractMethodOne() is implemented      }        //This class must implement second abstract method also,      //otherwise, this class has to be declared as abstract        void abstractMethodTwo()      {          //abstractMethodTwo() is also implemented.          //No need to declare this class as abstract      }  } |

* Inside abstract class, we can keep any number of constructors. If you are not keeping any constructors, then compiler will keep default constructor.

[?](http://javaconceptoftheday.com/abstraction-in-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | abstract class AbstractClass  {      AbstractClass()      {          //First Constructor      }        AbstractClass(int i)      {          //Second Constructor      }        abstract void abstractMethodOne();  //Abstract Method  } |

* Abstract methods can not be private. Because, abstract methods must be implemented somehow in the sub classes. If you declare them as private, then you can’t use them outside the class.

[?](http://javaconceptoftheday.com/abstraction-in-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | abstract class AbstractClass  {      private abstract void abstractMethodOne();      //Compile time error, abstract method can not be private.  } |

* Constructors and fields can not be declared as abstract.

[?](http://javaconceptoftheday.com/abstraction-in-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | abstract class AbstractClass  {      abstract int i;      //Compile time error, field can not be abstract        abstract AbstractClass()      {          //Compile time error, constructor can not be abstract      }  } |

* Abstract methods can not be static.

[?](http://javaconceptoftheday.com/abstraction-in-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | abstract class AbstractClass  {      static abstract void abstractMethod();      //Compile time error, abstract methods can not be static  } |

## Java Interview Questions On Abstract Class :

**1) Abstract class must have only abstract methods. True or false?**

False. Abstract methods can also have concrete methods.

**2) Is it compulsory for a class which is declared as abstract to have at least one abstract method?**

Not necessarily. Abstract class may or may not have abstract methods.

**3) Can we use “abstract” keyword with constructor, Instance Initialization Block and Static Initialization Block?**

No. Constructor, Static Initialization Block, Instance Initialization Block and variables can not be abstract.

**4) Why final and abstract can not be used at a time?**

Because, final and abstract are totally opposite in nature. A final class or method can not be modified further where as abstract class or method must be modified further. “final” keyword is used to denote that a class or method does not need further improvements. “abstract” keyword is used to denote that a class or method needs further improvements.

**5) Can we instantiate a class which does not have even a single abstract methods but declared as abstract?**

No, We can’t instantiate a class once it is declared as abstract even though it does not have abstract methods.

**6) Can we declare abstract methods as private? Justify your answer?**

No. Abstract methods can not be private. If abstract methods are allowed to be private, then they will not be inherited to sub class and will not get enhanced.

**7) We can’t instantiate an abstract class. Then why constructors are allowed in abstract class?**

It is because, we can’t create objects to abstract classes but we can create objects to their sub classes. From sub class constructor, there will be an implicit call to super class constructor. That’s why abstract classes should have constructors. Even if you don’t write constructor for your abstract class, compiler will keep default constructor.

**8) Can we declare abstract methods as static?**

No, abstract methods can not be static.

**9) Can a class contain an abstract class as a member?**

Yes, a class can have abstract class as it’s member.

**10) Abstract classes can be nested. True or false?**

True. Abstract classes can be nested i.e an abstract class can have another abstract class as it’s member.

**11) Can we declare abstract methods as synchronized?**

No, abstract methods can not be declared as synchronized. But methods which override abstract methods can be declared as synchronized.

**12) Can we declare local inner class as abstract?**

Yes. Local inner class can be abstract.

**13) Can abstract method declaration include throws clause?**

Yes. Abstract methods can be declared with throws clause.

[Java Practice Coding Questions On Abstract Classes](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/)

**1) Below class ABC doesn’t have even a single abstract method, but it has been declared as abstract. Is it correct?**

|  |
| --- |
| abstract class ABC  {      void firstMethod()      {          System.out.println("First Method");      }        void secondMethod()      {          System.out.println("Second Method");      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/#collapse1)

**Answer :**  
Yes, it is correct. abstract classes may or may not have abstract methods.

**2) Why the below class is showing compilation error?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | abstract class AbstractClass  {      abstract void abstractMethod()      {          System.out.println("First Method");      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/#collapse2)

**Answer :**  
Because, abstract methods must not have a body.

**3) Which class is instantiable? Class A or Class B?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | abstract class A  {    }    class B extends A  {    } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/#collapse3)

**Answer :**  
Class B.

**4) Below code snippet is showing compilation error? Can you suggest the corrections?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | abstract class A  {      abstract int add(int a, int b);  }    class B extends A  {    } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/#collapse4)

**Answer :**  
Class B must implement inherited abstract method A.add() or else it must be declared as abstract.

**5) Is the following program written correctly? If yes, what value “result” variable will hold if you run the program?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | abstract class Calculate  {      abstract int add(int a, int b);  }    public class MainClass  {      public static void main(String[] args)      {          int result = new Calculate()          {              @Override              int add(int a, int b)              {                  return a+b;              }          }.add(11010, 022011);      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/#collapse5)

**Answer :**  
Yes, program is written correctly. result = 20235.

**6) Can we write explicit constructors in an abstract class?**

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/#collapse6)

**Answer :**  
Yes. abstract classes can have any number of constructors.

**7) Can you identify the error in the below code?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/)

|  |  |
| --- | --- |
| 1  2  3  4 | abstract class AbstractClass  {      private abstract int abstractMethod();  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/#collapse7)

**Answer :**  
abstract methods can’t be private.

**8) Can we declare protected methods in an interface?**

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/#collapse8)

**Answer :**  
No. only public methods are allowed in an interface.

**9) What will be the output of the following program?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | abstract class A  {      abstract void firstMethod();        void secondMethod()      {          System.out.println("SECOND");            firstMethod();      }  }    abstract class B extends A  {      @Override      void firstMethod()      {          System.out.println("FIRST");            thirdMethod();      }        abstract void thirdMethod();  }    class C extends B  {      @Override      void thirdMethod()      {          System.out.println("THIRD");      }  }    public class MainClass  {      public static void main(String[] args)      {          C c = new C();            c.firstMethod();            c.secondMethod();            c.thirdMethod();      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/#collapse9)

**Answer :**  
FIRST  
THIRD  
SECOND  
FIRST  
THIRD  
THIRD

**10) What will be the output of the below program?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | abstract class X  {      public X()      {          System.out.println("ONE");      }        abstract void abstractMethod();  }    class Y extends X  {      public Y()      {          System.out.println("TWO");      }        @Override      void abstractMethod()      {          System.out.println("THREE");      }  }    public class MainClass  {      public static void main(String[] args)      {          X x = new Y();            x.abstractMethod();      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/#collapse10)

**Answer :**  
ONE  
TWO  
THREE

**11) Can we declare abstract methods as static?**

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/#collapse11)

**Answer :**  
No. abstract methods can’t be static.

**12) Is the below program written correctly? If yes, what will be the output?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | abstract class A  {      {          System.out.println("AAA");      }  }    abstract class B extends A  {      {          System.out.println("BBB");      }  }    class C extends B  {      {          System.out.println("CCC");      }  }    public class MainClass  {      public static void main(String[] args)      {          C c = new C();      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/#collapse12)

**Answer :**  
Yes, program is written correctly. Output will be,  
AAA  
BBB  
CCC

**13) What will be the output of the following program?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | abstract class A  {      abstract int firstMethod(int i);        abstract int secondMethod(int i);        int thirdMethod(int i)      {          return secondMethod(++i);      }  }    abstract class B extends A  {      @Override      int secondMethod(int i)      {          return firstMethod(++i);      }  }    class C extends B  {      @Override      int firstMethod(int i)      {          return ++i;      }  }    public class MainClass  {      public static void main(String[] args)      {          C c = new C();            System.out.println(c.thirdMethod(121121));      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/#collapse13)

**Answer :**  
121124

**14) Can we keep static initialization blocks inside an abstract class?**

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/#collapse14)

**Answer :**  
Yes. Abstract classes can have static initialization blocks as well as instance initialization blocks.

**15) Is the below program written correctly? If yes, what will be the output?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | abstract class XYZ  {      {          System.out.println(1);      }        public XYZ()      {          System.out.println(2);            abstractMethod();      }        abstract void abstractMethod();  }    class PQR extends XYZ  {      {          System.out.println(3);      }        public PQR()      {          System.out.println(4);      }        @Override      void abstractMethod()      {          System.out.println(5);      }  }    public class MainClass  {      public static void main(String[] args)      {          PQR pqr = new PQR();      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/#collapse15)

**Answer :**  
Yes, program is written correctly. Output will be,  
1  
2  
5  
3  
4

**16) Can you identify the error in the below code?**

|  |
| --- |
| class X  {      public X()      {          System.out.println("Constructor One");      }        abstract X(int i)      {          System.out.println("Constructor Two");      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/#collapse16)

**Answer :**  
Constructors can’t be abstract.

**17) Abstract methods can be declared as final. True or False?**

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/#collapse17)

**Answer :**  
False. Abstract methods can’t be final.

**18) Is the below code written correctly?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class X  {      abstract class Y      {          class Z          {            }      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/#collapse18)

**Answer :**  
Yes, code is written correctly.

**19) What will be the output of the following program?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | class ClassOne  {      int methodOne(int i, int j)      {          return i++ + ++j - ++i - j++;      }  }    abstract class ClassTwo extends ClassOne  {      abstract int methodOne(int i, int j, int k);        @Override      int methodOne(int i, int j)      {          return methodOne(i, j, i+j);      }  }    class ClassThree extends ClassTwo  {      @Override      int methodOne(int i, int j, int k)      {          return --i - j-- + ++k - i++ + ++j - k--;      }  }    public class MainClass  {      public static void main(String[] args)      {          ClassOne one = new ClassOne();            ClassThree three = new ClassThree();            System.out.println(three.methodOne(one.methodOne(10101, 20202), one.methodOne(20202, 10101)));      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/#collapse19)

**Answer :**  
0

**20) Is the below code written correctly?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class A  {      void methodOfA()      {          abstract class B          {            }      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/2/#collapse20)

**Answer :**  
Yes, code is written correctly. Local inner classes can be abstract.

**21) Can we declare abstract method with throws clause?**

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse21)

**Answer :**  
Yes, abstract methods can be declared with throws clause.

**22) What will be the output of the following program?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | abstract class A  {      int i = 111, j = 222;        abstract void methodOne();        abstract void methodTwo();  }    abstract class B extends A  {      @Override      void methodOne()      {          System.out.println(i);            System.out.println(j);            i = ++i;            j = --j;      }  }    class C extends B  {      @Override      void methodTwo()      {          System.out.println(i);            System.out.println(j);            i = i++;            j = j--;      }  }    public class MainClass  {      public static void main(String[] args)      {          C c = new C();            c.methodOne();            c.methodTwo();            System.out.println(c.i);            System.out.println(c.j);      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse22)

**Answer :**  
111  
222  
112  
221  
112  
221

**23) Which of the following shows 100% abstractness?  
a) Abstract Classes b) Interfaces**

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse23)

**Answer :**  
b) Interfaces

Abstract classes are not 100% abstract because of concrete constructors.

**24) Can you identify the error in the below code?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/)

|  |  |
| --- | --- |
| 1  2  3  4 | abstract class A  {      synchronized abstract void method();  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse24)

**Answer :**  
Abstract methods can not be declared as synchronized.

**25) What will be the output of the following program?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | abstract class X  {      int i = 111;        int methodX()      {          return methodX(i);      }        abstract int methodX(int i);  }    class Y extends X  {      @Override      int methodX(int i)      {          return ++i + i++;      }  }    public class MainClass  {      public static void main(String[] args)      {          Y y = new Y();            System.out.println(y.methodX());      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse25)

**Answer :**  
224

**26) Is the below code written correctly?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class A  {      abstract class B      {          class C          {              abstract class D              {                  abstract void method();              }          }      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse26)

**Answer :**  
Yes, code is correct.

**27) Write a code which implements abstract method “methodY()” of class Y in the below code?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class X  {      abstract static class Y      {          abstract void methodY();      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse27)

**Answer :**

|  |
| --- |
| class Z extends X.Y  {      @Override      void methodY()      {          System.out.println("methodY implementation");      }  } |

**28) Can we instantiate a class which has only concrete methods but declared as abstract?**

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse28)

**Answer :**  
No. We can’t instantiate an abstract class even though it has only concrete methods.

**29) What will be the output of the following program?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | abstract class ABC  {      abstract void methodOne();  }    abstract class XYZ extends ABC  {      int i;        @Override      void methodOne()      {          methodOne(i \*= i);      }        abstract void methodOne(int i);  }    class PQR extends XYZ  {      public PQR(int i)      {          this.i = i;      }        @Override      void methodOne(int i)      {          System.out.println(i++ \* ++i);      }  }    public class MainClass  {      public static void main(String[] args)      {          PQR pqr = new PQR(1);            pqr.methodOne();      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse29)

**Answer :**  
3

**30) Is the following program written correctly? If yes, what will be the output?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | abstract class A  {      {          methodA();      }        abstract void methodA();  }    class B extends A  {      @Override      void methodA()      {          System.out.println("methodA");      }  }    public class MainClass  {      public static void main(String[] args)      {          new B();      }  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse30)

**Answer :**  
Yes, program is correct. Output will be,  
methodA

**31) One class has a method with two overloaded forms. One form is abstract and another one is concrete. Is it possible in java?**

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse31)

**Answer :**  
Yes. Overloaded methods can be abstract or concrete.

**32) Which line in the below code shows compile time error?**

[?](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | abstract class X  {      public X()      {          methodX();      }        static      {          methodX();      }        abstract void methodX();  } |

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse32)

**Answer :**  
Line 10. Because, you can’t make a static reference to non-static method.

**33) You know that we can’t create objects to an abstract class. Then, why it is allowed to have constructors in an abstract class?**

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse33)

**Answer :**  
Because, these constructors will be called in constructor chaining.

**34) You know that abstract methods can’t be static. Can we declare abstract inner classes as static?**

[**View Answer**](http://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/3/#collapse34)

**Answer :**  
Yes, abstract inner classes can be static.