#For Google

**//how to access global variable in c when local variable with same name is present**

**An abstract class in Java is one that is declared with the abstract keyword. It may have both abstract and non-abstract methods(methods with bodies). An abstract is a java modifier applicable for classes and methods in java but*not for Variables*.**

**Illustration: Abstract class**

**abstract class Shape**

**{**

**int color;**

**// An abstract function**

**abstract void draw();**

**}**

**In java, the following some *important observations*about abstract classes are as follows:**

1. **An instance of an abstract class can not be created.**
2. **Constructors are allowed.**
3. **We can have an abstract class without any abstract method.**
4. **There can be a final method in abstract class but any abstract method in class(abstract class) can not be declared as final  or in simpler terms final method can not be abstract itself as it will yield an error: “Illegal combination of modifiers: abstract and final”**
5. **We can define static methods in an abstract class**
6. **We can use the abstract keyword for declaring *top-level classes (Outer class) as well as inner classes* as abstract**
7. **If a class contains at least one abstract method then compulsory should declare a class as abstract**
8. **If the Child class is unable to provide implementation to all abstract methods of the Parent class then we should declare that Child class as abstract so that the next level Child class should provide implementation to the remaining abstract method**

**Let us elaborate on these observations and do justify them with help of clean java programs as follows.**

**Observation 1: In Java, just like in C++ an instance of an abstract class cannot be created, we can have references to abstract class type though. It is as shown below via the clean java program.**

**Example**

* **Java**

|  |
| --- |
| **// Java Program to Illustrate**  **// that an instance of Abstract**  **// Class can not be created**    **// Class 1**  **// Abstract class**  **abstract class Base {**  **abstract void fun();**  **}**    **// Class 2**  **class Derived extends Base {**  **void fun()**  **{**  **System.out.println("Derived fun() called");**  **}**  **}**    **// Class 3**  **// Main class**  **class Main {**    **// Main driver method**  **public static void main(String args[])**  **{**    **// Uncommenting the following line will cause**  **// compiler error as the line tries to create an**  **// instance of abstract class. Base b = new Base();**    **// We can have references of Base type.**  **Base b = new Derived();**  **b.fun();**  **}**  **}** |

**Output**

**Derived fun() called**

**Observation 2: Like C++, an abstract class can contain constructors in Java. And a constructor of an abstract class is called when an instance of an inherited class is created. It is as shown in the program below as follows:**

**Example**

* **Java**

|  |
| --- |
| **// Java Program to Illustrate Abstract Class**  **// Can contain Constructors**    **// Class 1**  **// Abstract class**  **abstract class Base {**    **// Constructor of class 1**  **Base()**  **{**  **// Print statement**  **System.out.println("Base Constructor Called");**  **}**    **// Abstract method inside class1**  **abstract void fun();**  **}**    **// Class 2**  **class Derived extends Base {**    **// Constructor of class2**  **Derived()**  **{**  **System.out.println("Derived Constructor Called");**  **}**    **// Method of class2**  **void fun()**  **{**  **System.out.println("Derived fun() called");**  **}**  **}**    **// Class 3**  **// Main class**  **class GFG {**    **// Main driver method**  **public static void main(String args[])**  **{**  **// Creating object of class 2**  **// inside main() method**  **Derived d = new Derived();**  **d.fun();**  **}**  **}** |

**Output**

**Base Constructor Called**

**Derived Constructor Called**

**Derived fun() called**

**Observation 3: In Java, we can have *an abstract class without any abstract method*. This allows us to *create classes that cannot be instantiated but can only be inherited*. It is as shown below as follows with help of a clean java program.**

**Example**

* **Java**

|  |
| --- |
| **// Java Program to illustrate Abstract class**  **// Without any abstract method**    **// Class 1**  **// An abstract class without any abstract method**  **abstract class Base {**    **// Demo method. This is not an abstract method.**  **void fun()**  **{**  **// Print message if class 1 function is called**  **System.out.println("Function of Base class is called");**  **}**  **}**    **// Class 2**  **class Derived extends Base {**  **//This class only inherits the Base class methods and properties**    **}**    **// Class 3**  **class Main {**    **// Main driver method**  **public static void main(String args[])**  **{**  **// Creating object of class 2**  **Derived d = new Derived();**    **// Calling function defined in class 1 inside main()**  **// with object of class 2 inside main() method**  **d.fun();**  **}**  **}** |

**Output**

**Function of Base class is called**

**Observation 4: *Abstract classes can also have final methods*(methods that cannot be overridden)**

**Example**

* **Java**

|  |
| --- |
| **// Java Program to Illustrate Abstract classes**  **// Can also have Final Methods**    **// Class 1**  **// Abstract class**  **abstract class Base {**    **final void fun()**  **{**  **System.out.println("Base fun() called");**  **}**  **}**    **// Class 2**  **class Derived extends Base {**    **}**    **// Class 3**  **// Main class**  **class GFG {**    **// Main driver method**  **public static void main(String args[])**  **{**  **{**  **// Creating object of abstract class**    **Base b = new Derived();**  **// Calling method on object created above**  **// inside main method**    **b.fun();**  **}**  **}**  **}** |

**Output:**

**Base fun() called**

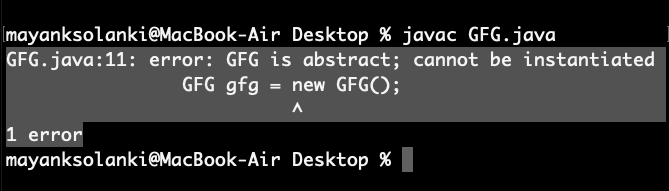
**Observation 5: For any abstract java class we are not allowed to create an object i.e., for abstract class instantiation is not possible.**

**Example**

* **Java**

|  |
| --- |
| **// Java Program to Illustrate Abstract Class**    **// Main class**  **// An abstract class**  **abstract class GFG {**    **// Main driver method**  **public static void main(String args[])**  **{**    **// Trying to create an object**  **GFG gfg = new GFG();**  **}**  **}** |

**Output:**

****

**Observation 6: Similar to the interface *we can define static methods in an abstract class* that*can be called independently without an object.***

**Example**

* **Java**

|  |
| --- |
| **// Java Program to Illustrate**  **// Static Methods in Abstract**  **// Class Can be called Independently**    **// Class 1**  **// Abstract class**  **abstract class Helper {**    **// Abstract method**  **static void demofun()**  **{**    **// Print statement**  **System.out.println("Geeks for Geeks");**  **}**  **}**    **// Class 2**  **// Main class extending Helper class**  **public class GFG extends Helper {**    **// Main driver method**  **public static void main(String[] args)**  **{**    **// Calling method inside main()**  **// as defined in above class**  **Helper.demofun();**  **}**  **}** |

**Output**

**Geeks for Geeks**

**Observation 7: We can use the abstract keyword for declaring top-level classes (Outer class) as well as inner classes as abstract**

* **Java**

|  |
| --- |
| **import java.io.\*;**    **abstract class B {**  **// declaring inner class as abstract with abstract**  **// method**  **abstract class C {**  **abstract void myAbstractMethod();**  **}**  **}**  **class D extends B {**  **class E extends C {**  **// implementing the abstract method**  **void myAbstractMethod()**  **{**  **System.out.println(**  **"Inside abstract method implementation");**  **}**  **}**  **}**    **public class Main {**    **public static void main(String args[])**  **{**  **// Instantiating the outer class**  **D outer = new D();**    **// Instantiating the inner class**  **D.E inner = outer.new E();**  **inner.myAbstractMethod();**  **}**  **}** |

**Output:**

**Inside abstract method implementation**

**Observation 8: If a class contains at least one abstract method then compulsory we should declare the class as abstract otherwise we will get a compile-time error because If a class contains at least one abstract method then, implementation is not complete for that class, and hence it is not recommended to create an object so in order to restrict object creation for such partial classes we use abstract keyword.**

* **Java**

|  |
| --- |
| **/\*package whatever //do not write package name here \*/**    **import java.io.\*;**    **// here if we remove the abstract**  **// keyword then we will get compile**  **// time error due to abstract method**  **abstract class Demo {**  **abstract void m1();**  **}**    **class Child extends Demo {**  **public void m1()**  **{**  **System.out.print("Hello");**  **}**  **}**  **class GFG {**  **public static void main(String[] args)**  **{**  **Child c = new Child();**  **c.m1();**  **}**  **}** |

**Output:**

**Hello**

**Observation 9: If the Child class is unable to provide implementation to all abstract methods of the Parent class then we should declare that Child class as abstract so that the next level Child class should provide implementation to the remaining abstract method.**

* **Java**

|  |
| --- |
| **/\*package whatever //do not write package name here \*/**    **import java.io.\*;**  **abstract class Demo {**  **abstract void m1();**  **abstract void m2();**  **abstract void m3();**  **}**  **abstract class FirstChild extends Demo {**  **public void m1() {**  **System.out.println("Inside m1");**  **}**  **}**    **class SecondChild extends FirstChild {**  **public void m2() {**  **System.out.println("Inside m2");**  **}**  **public void m3() {**  **System.out.println("Inside m3");**  **}**  **}**  **class GFG {**  **public static void main(String[] args)**  **{**  **// if we remove the abstract keyword from FirstChild**  **// Class and uncommented below obj creation for**  **// FirstChild then it will throw**  **// compile time error as did't override all the**  **// abstract methods**    **// FirstChild f=new FirstChild();**  **// f.m1();**    **SecondChild s = new SecondChild();**  **s.m1();**  **s.m2();**  **s.m3();**  **}**  **}** |

**Output:**

**Inside m1**

**Inside m2**

**Inside m3**

**In C++, if a class has at least one**[**pure virtual function**](https://www.geeksforgeeks.org/pure-virtual-functions-and-abstract-classes/)**, then the class becomes abstract. Unlike C++, in Java, a separate keyword abstract is used to make a class abstract.**

**https://youtu.be/DWmpxZ59JW0**

**Must Read:**

* [**Difference between Abstract class and Interface in Java**](https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/)
* [**Difference between Abstract class and Abstract Methods**](https://www.geeksforgeeks.org/difference-between-abstract-class-and-abstract-method-in-java/)
* [**Constructors in Java Abstract Class**](https://www.geeksforgeeks.org/constructor-in-java-abstract-class/)

**Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.**