|  |  |  |
| --- | --- | --- |
| 1. | **What is abstract class or abstract method?**   1. An abstract class is also known as un-implemented/declared/abstract method. 2. Abstract classes are classes that contain zero or more abstract methods. An abstract method is a method that is declared, but contains no implementation. 3. Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods. 4. If a class contains minimum 1 abstract method then the class should be declared as an abstract class. 5. For an abstract class. object can’t be created. 6. For an abstract class, a reference variable can be created i.e. For datatype purpose you can use abstract class to specify datatype. 7. abstract class identifier can be used as argument datatype as well as method return type. 8. class file is generating even for an abstract class also. 9. abstract methods should be implemented in the subclass otherwise, the subclass has to be declared as an abstract. i.e. If a subclass is inherited with abstract methods from the superclass then it should be implemented in subclass to be concrete/implemented otherwise subclass should also be declared as abstract class. 10. It is possible to develop multiple abstract method inside an abstract class. 11. We can develop any defined methods inside an abstract class. | |
| 2. | **When to use Abstract Methods and abstract class?**   1. Abstract methods are usually declared where two or more subclasses are expected to do similar thing in different ways through different implementations. These subclasses extend the same Abstract class and provide different implementations for the abstract methods. 2. Abstract classes are used to define generic types of behaviors and use its subclasses to provide implementation details of the abstract class. | |
| 3. | **What do you mean by concrete methods?**  If {, } brace both are there for a method even if statements is there or not it is called implemented/ defined/ready-to-use/concrete method. | |
| 4. | **Can a method be abstract as well as have implementation?**  No. If a method is an abstract at the same time it cannot have implementation. If both are present for same method on compilation you will get compile-time error. | |
| 5. | **What happens if a class containing abstract method is not declared as an abstract class?**  Compile-time error. If a class contains abstract method and it’s not declared as an abstract class compiler gives error specifying to create class as abstract. | |
| 6. | **Can a class without any abstract method be declared as an abstract class?**  Yes, definitely. We can declare any class as an abstract class. | |
| 7. | **When should a class without any abstract method be declared as abstract class?**  When we don’t want a class to be instantiated/ class object created then that class should be declared as an abstract class.  i.e. When a class without any abstract method is declared as abstract class then compiler won’t allow you to create object of that class.  **Note:** If a class is not containing any abstract method compiler doesn’t force to declare class as abstract but if class is abstract class then compiler doesn’t allow to create object. | |
| 8. | **Is it possible to create object of an abstract class?**  No, abstract class object cannot be created. | |
| 9. | **When constructor does execute?**  While creating object to current class or object to any of its subclass. | |
| 10. | **Can an abstract class have constructor?**  Yes, definitely. An abstract class can have constructor. It will execute while creating object to concrete subclass the superclass constructor(abstract class) will execute. | |
| 11. | **When does abstract class constructor will execute?**  While creating object to any subclass which is concrete. In its concrete subclass constructor through super calling statement by programmer or placed by compiler the abstract class constructor will be executing. | |
| 12. | **Why constructor can’t be static?**  Constructors can’t be static as it is for only object creation, where non-static members are involved, non-static members are loading while object is created. | |
| 13. | **Why constructor can’t be abstract?**  As the constructor is not inheriting to subclass and as it is not inheriting to subclass, where will it be implemented? So constructor has to be implemented in the same class for which it is. Thus, constructor can’t be abstract. | |
| 14. | **Can a class be 100% abstract?**  No. Even though, a class is containing all abstract methods still it is not 100% abstract as it contains a constructor as well as several inherited concrete methods from java.lang.Object. | |
| 15. | **Is it possible to run abstract class, if it contains properly defined main() method?**  Yes, absolutely possible. For abstract class object can’t be created but we can run a class with main () method. To run main () method, because main() is static so object creation is not required. | |
| 16. | **Explain interface briefly.**  To achieve 100% abstractness we develop interface.  By default, an interface is 100% abstract. So ‘abstract’ keyword is optional.  For an interface, .class file will be generating.  interface can become member of .java file.  For interface an object can’t be created.  For an interface a reference variable can be created as well as it can be as a method argument, method returntype. | |
| 17. | **In this exercise we want you to take an abstract class which we have defined for you and develop two classes. The abstract class represents the basic building block for employees in a personnel database. The code is shown below:**  abstract class Employee {  private String name, address;  protected int basicSalary;  public String getName(){  return name;  }  public String getAddress(){  return address;  }  public int getBasicSalary(){  return basicSalary;  }  public void setAddress(String add){  address = add;  }  public void setName(String nm){  name = nm;  }  public void setBasicSalary(int sal){  basicSalary = sal;  }  public abstract int getMonthlySalary();  }  The class contains three instance variables which hold the name, address and basic yearly salary of an employee.  Create 2 subclasses to Employee class :  NormalEmployee:  This class should have a single method which calculates the monthly salary for an employee.  BonusEmployee  This class describes an employee who has a monthly bonus added to their monthly salary. | |
| 1. | **What different types of members a java file can contain?**  class  abstract class  interface  annotation  enum  -For each of the above members a separate class file will be generated on compilation. | |
| 2. | **By default, the methods in interface are \_\_\_\_\_\_\_\_\_\_\_\_\_\_.**  public abstract. | |
| 3. | **What kind of methods a class can contain?**  A class can contain i) method with declaration, ii) method with definition.  To differentiate between 2 types of methods ‘abstract’ keyword is required for all the declared methods of a class. | |
| 4. | **What kind of methods an interface can contain?**  An interface can contain only one type of methods i.e. method with declaration only. An interface methods are all abstract methods. | |
| 5. | **Is it compulsory to specify ‘abstract’ keyword before methods in interface?**  No. Even if a method doesn’t have ‘abstract’ keyword still it will compile fine. Compiler will keep the ‘abstract’ keyword if programmer has not specified.  ‘abstract’ keyword is optional in an interface.  [**Note :** In class ‘abstract’ keyword is compulsory.] | |
| 6. | **Describe kind of members does an interface contains.**  Every member in an interface is ‘public’ by default. Even though, programmer doesn’t specify ‘public’ keyword all members are public. An interface member cannot be declared with private or protected access level explicitly. Either specify access level as public or don’t specify.  Inside an interface 2 types of members are possible:   1. constants 2. abstract methods | |
| 7. | **What kind of implementations, an interface cannot contain?Why?**  In an interface, following kind of implementations are not possible:   * Method with body * Constructor with body * IIB * SIB   Because interface are meant for achieving 100% abstractness. | |
| 8. | **Why an interface can’t run?**  An interface can’t be run, as in interface we can have only abstract methods. To run any .class file it should contain implemented public static void main(String[] args). | |
| 9. | **Is it a valid method/constructor in an interface T?**   1. public abstract void test1();🡪 Yes, as method is public abstract and is ended with ; 2. public int test2(String a); 🡪Yes, as method is public and is just containing declaration 3. float test3(int x); 🡪 Yes, as method is treated as ‘public abstract’, by default. 4. private void test()🡪No,a method in an interface can be declared with public access level or no access level. 5. voiddoTest(){ } 🡪 No, as an interface can contain only abstract methods. 6. public T(){}🡪No, constructor can’t be defined inside interface as it is 100% abstract. 7. public T(int x); 🡪No, constructor can’t be declared in an interface. As constructor is not inherited to subclass that implements an interface. As well as, constructor executes while creating object but for an interface you can’t create an object. | |
| 10. | **What happens if an interface contains an implemented method or constructor or IIB or SIB?**  Compile-time error. As inside an interface definition blocks are not permitted.  Implemented method, constructor, IIB, SIB all falls under definition blocks. | |
| 11. | **Can an interface Test contain an interface A, a class B or an abstract class C inside it?**  i.e.  interface Test{  interface A  {  }  class B  {  }  abstract class C  {  }  }  Yes. As an interface, a class or an abstract class is not a definition block though they have an opening and closing brace.  These are treated as inner members within an interface Test.  On compilation of the java file containing the above interface 4 .class files will be generated:  Test.class  Test$A.class  Test$B.class  Test$C.class | |
| 12. | **Using, which keyword inheritance can be achieved in java?**  extends and implements | |
| 13. | **If a subclass implements an interface containing 4 methods and is implementing only 2 methods of those, Can I create an instance(object) of the subclass?**  No. As subclass is implementing an interface it has 4 inherited abstract methods but only 2 methods are implemented and 2 methods are still abstract. So, subclass will be an abstract class thus instantiation(means object creation) is not possible. | |
| 14. | **State default scope of an interface.**  The default scope of an interface is default. | |
| 15. | **While implementing a method of an interface in a subclass what should be followed ?**  When an interface method is implemented in a subclass. While implementing a method, same signature, same return type, method should be public for proper implementation. | |
| 16. | **What is the access modifier for a method inherited from interface, an abstract class?**  For method inherited from an interface access modifier will be public.  For method inherited from an abstract class access modifier will be same as the one for method in abstract class or wider access level.  Eg method is protected in abstract class then in subclass method can be protected or public. | |
| 17. | **Can a class simultaneously extend a class as well as implement an interface?**  Yes. But while doing so order should be taken care. i.e. first extends and then implements. | |
| 18. | **Can a class extend multiple classes?**  No a class can extend only one class at a time. Multiple inheritance through classes is not possible in java.  Multiple inheritance is creating a new class from existing multiple classes  i.e. creating class C from class A and class B. | |
| 19. | **Does java support multiple inheritances through interfaces?**  Yes. A class can implement any no. of interfaces.  class ClassName implements interface1,interface2,…,interfaceN  {  //implement methods from n interfaces  } | |
| 20. | **What happens if we try to have multiple inheritance through class in java?**  On trying, multiple inheritance through class will give compile-time error. | |
| 1. | | **What happens if a subclass C is implementing interface A and extending class B as follows:**  **class C implements A extends B ?**  Compile time error as order of extends and implements has to be followed. |
| 2. | | **What happens if an interface A and a class B both contains same named method with same signature public void test()? And a subclass C is extending class B and implementing interface A.**  class C will be getting an inherited abstract method from A but it gets automatically implemented as it also extends B which is containing implemented void test(). |
| 3. | | **How inheritance is possible between 2 interfaces, 2 classes, and interface- class?**  To develop an interface using an interface use ‘extends’ keyword  To develop a class using a class use ‘extends’ keyword  To develop a class using an interface use ‘implements’ keyword. |
| 4. | | **How many classes a class can extend?**  A class can extend maximum 1 class.  Eg.:  class A{}  class B extends A{} |
| 5. | | **How many interfaces a class can implement at a time?**  A class can implement any no. of interfaces.  Eg:  interface A{}  interface C{}  interface D{}  class B implements A,C,D{} |
| 6. | | **How many interfaces an interface can extends?**  An interface can extend any no. of interfaces.  Eg:  interface A{}  interface C{}  interface D{}  interface B extends A,C,D{} |
| 7. | | **If an abstract class A contains a method as *public abstract void test()* and in a concrete subclass B void test(boolean b){ } is defined, will it compile successfully?**  It will result in compile time error. |
| 8. | | **Define : Method Overloading**  Developing multiple methods with same name but different signature is called as method overloading.  Eg:  void test()  int test(int x)  int test (float a, int b)  above 3 methods are having same name i.e. test but signatures are different.  Method signature is methodName(argument); i.e. argument is total no. of argument, datatype of arguments, order of arguments should be different.  While menthod is overloaded they may have same return type or different return type, same or different access level. |
| 9. | | **How method overloading cannot be done?** |
|  | | Keeping same signature of method, method overloading can’t be possible.  i.e.  1. Just changing the return type of the method. If the return type of the method is the only thing  changed, then this will result in a compiler error.  2. Changing just the name of the method parameters, but not changing the parameter types. If the name of the method parameter is the only thing changed then this will also result in a compiler error. |
| 10. | | **How method duplication is identified by the compiler?**  Method duplication is identified through their signature. |
| 11. | | **Identify valid/invalid case of method overloading:**   1. int mymethod(int a, int b, float c)   int mymethod(int var1, int var2, float var3)  Compile time error. Argument lists are exactly same. Both methods are having same number, data types and same sequence of data types in arguments.   1. int mymethod(int a, int b)   int mymethod(float var1, float var2)  Valid case for overloading. Here data types of arguments are different.   1. int mymethod(int a, int b)   int mymethod(int num)  Valid case for overloading. Here number of arguments are different.   1. float mymethod(int a, float b)   float mymethod(float var1, int var2)  Valid case for overloading. Sequence of the data types are different, first method is having (int, float) and second is having (float, int).   1. int mymethod(int a, int b)   float mymethod(int var1, int var2)  Compile time error. Argument lists are exactly same. Even though return type of methods are different, it is not a valid case. Since return type of method doesn’t matter while overloading a method. |
| 12. | | **Why method overloading is not possible by changing the return type in java?**  In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity.  Eg:  class Calculation3{  int sum(int a,int b){System.out.println(a+b);}  double sum(int a,int b){System.out.println(a+b);}  public static void main(String args[]){  Calculation3 obj=new Calculation3();  int result=obj.sum(20,20); //Compile Time Error  }  }  **Reason :** int result=obj.sum(20,20); //Here how can java determine which sum() method should be called |
| 13. | | **Can we overload main() method?**  Yes. You can define any no. of main() methods if each are having different signatures.  Eg:  class Overloading1{  public static void main(int a){  System.out.println(a);  }  public static void main(String args[]){  System.out.println("main() method invoked");  main(10);  }  }  Output: main() method invoked  10 |
| 14. | | **While overloading the methods, is access level, abstract, static, return type of importance?**  No. While overloading the method only signature is taken care of. |
| 15. | | **Define: Method Overriding**  If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.  When a implemented method of super class is inherited to subclass and is re-implemented in subclass with same signature then it is called as method overriding.  Rules :   1. Argument List :   The argument list of overriding method must be same as that of the method in parent class. The data types of the arguments and their sequence should be maintained as it is in the overriding method.   1. Access Modifier:   The Access Modifier of the overriding method (method of subclass) cannot be more restrictive than the overridden method of parent class. For e.g. if the Access Modifier of base class method is public then the overriding method (child class method ) cannot have private, protected and default Access modifier as all of the three are more restrictive than public.   1. private and final methods cannot be overridden as they are local to the class. 2. If a method is with static in super class then in subclass it should be overridden with static in the subclass too. 3. Return type:   In case of void and primitive return type, while overriding /implementing the method of superclass/interface, return type should be same.  If at all super class method’s return type is derived datatype while overriding in the subclass you can go for same derived datatype or any of its subclass.   1. super keyword in overriding   super keyword is used for calling the parent class method/constructor. super.methodname() calling the specified method of base class while super() calls the constructor of base class. |
| 16. | | **Describe the concept of covariant.**  If a superclass method with derived datatype as its return type, while overriding the same method its return type can be same derived datatype or subclass to it.  Eg.  class A{}  class B extends A{}  class C  {  A test(){}  }  class D extends C  {  B test(){}  } |
| 17. | | **What if while overriding a method, signature is same but return type in super class is void and subclass is different ?**  Compile time error. |
| 18. | | **What if while overriding a method with derived data type as return type in super class and in a subclass is overridden with narrowed derived data type?**  Compile time error. As it is violating co-variants.  Eg.  class A{}  class B extends A{}  class C  {  B test(){}  }  class D extends C  {  A test(){}  } |
| 19 | | **What happens if a method named public void test() in class A is overridden in subclass named B as protected void test()?**  Compile time error. As access level is narrower in overridden method |
| 20. | | **What happens if a method named void test() in a class A is overridden in subclass named B as protected void test()?**  Overridden method is valid as its access level is wider than the method in superclass. |
| 21 | | **If a method of an interface void test() is implemented in a subclass as void test(){}.Is it valid?**  No. As method in interface is by default public access level but in subclass it is implemented with default access level. |
| 22. | | **Can an attribute have same name as classname?**  Yes. An attributename can be any name if it follows the rule for compiler. |
| 23. | | **Can a method have same name as classname?**  Yes. A method can have same name as classname. Method can have any name including classname but it should have return type. |
| 24. | | **Can a class contain a method and constructor with same name and signature?**  Yes. As there only one method and one constructor. |
| 25. | | **Why the methods of interface are public and abstract by default ?**  Interface methods are public since they should be available to third party vendors to provide implementation. They are abstract because their implementation is left for third party vendors. |
| 26. | | **What is the difference between an abstract class and an interface ?**   |  |  | | --- | --- | | **Abstract class** | **Interface** | | **An abstract class is written when there are some common features shared by all the objects.** | **An interface is written when all the features are implemented differently in different objects.** | | **When an abstract class is written, it is the duty of the programmer to provide sub classes to it.** | **An interface is written when the programmer wants to leave the implementation to the third party vendors.** | | **An abstract class contains some abstract methods and also some concrete methods.** | **An interface contains only abstract methods.** | | **An abstract class contain instance variables also.** | **An interface cannot contain instance variables. It contains only constants.** | | **All the abstract methods of the abstract class should be implemented in its sub classes.** | **All the (abstract) methods of the interface should be implemented in its implementation classes.** | | **Abstract class is declared by using the keyword abstract.** | **Interface is declared using the keyword interface.** | |
| 27. | | **What is the difference between the following two statements.**  **1. import pack.Addition;**  **2. import pack.\*;**  In statement 1, only the Addition class of the package pack is imported into the program and in statement 2, all the classes and interfaces of the package pack are available to the program. |