Oracle

Exam 1z0-804

Java SE 7 Programmer II

Version: 8.0

[Total Questions: 150]

Question No : 1

Given:

```
class A {
    int a = 5;
    String doA() { return "a1 "; }
    protected static String doA2 () { return "a2 "; }
  }

class B extends A {
   int a = 7;
   String doA() { return "b1 "; }
   public static String doA2() { return "b2 "; }

   void go() {
      A myA = new B();
      System.out.print(myA.doA() + myA.doA2() + myA.a);
   }

   public static void main (String[] args) {
      new B().go();
   }
}
```

Which three values will appear in the output?

A. 5
B. 7
C. a1
D. a2
E. b1
F. b2

Answer: A,D,E

Explanation:

Staticmethod of base class is invoked >>

A myA = new B();

System.out.print(myA.doA() + myA.doA2() + myA.a);

class B String doA() { return "b1 "; }

class A protected static String doA2 () { return "a2 "; }
class B int a = 7;

Question No : 2

Given:

```
class Product {
    private int id;
    public Product (int id) {
      this.id = id;
    }
    public int hashCode() {
      return id + 42;
    }
    public boolean equals (Object obj) {
      return (this == obj) ? true : super.equals(obj);
    }
}

public class WareHouse {
    public static void main(String[] args) {
      Product p1 = new Product(10);
      Product p2 = new Product(10);
      Product p3 = new Product(20);
      System.out.print(p1.equals(p2) + " ");
      System.out.print(p1.equals(p3) );
    }
}
```

What is the result?

A. false false
B. true false
C. true true

D. Compilation fails
E. An exception is thrown at runtime

Answer: A

Explanation:
(this == obj) is the object implementation of equals() and therefore FALSE, if the reference points to variousobjectsand then the super.equals() is invoked, the object method equals() what still result in FALSEbetter override of equals() is to compare the attributes like:

```
public boolean equals (Object obj) {
if (obj != null){
Product p = (Product)obj;
return this.id == p.id;
}
return false;
}
```

Question No : 3

Given:

```
interface Rideable {
    String ride() ;
 }
 class Horse implements Rideable {
     String ride() { return "cantering "; }
 }

 class Icelandic extends Horse {
     String ride() { return "tolting "; }
 }

 public class Test1 {
     public static void main(String[] args) {

     Rideable r1 = new Icelandic();
     Rideable r2 = new Horse();
     Horse h1 = new Icelandic();

    System.out.println(r1.ride() + r2.ride() + h1.ride());

    }
 }
```

What is the result?

A. tolting cantering tolting
B. cantering cantering cantering
C. compilation fails
D. an exception is thrown at runtime

Answer: C

Explanation:

Compiler says: Cannot reduce the visibility of the inherited method from Rideable. müssen PUBLIC sein

public String ride() { return "cantering "; }

public String ride() { return "tolting "; }

if this is given then the result would be:

A : tolting cantering tolting

Question No : 4

Which four are syntactically correct?

A. package abc;
package def;
import Java.util . * ;
public class Test { }
B. package abc;
import Java.util.*;
import Java.util.regex.* ;
public class Test { }
C. package abc;
public class Test {}
import Java.util.* ;
D. import Java.util.*;
package abc;
public class Test {}
E. package abc;
import java.util. *;
public class Test{}
F. public class Test{}
package abc;
import java.util.*{}
G. import java.util.*;
public class Test{}
H. package abc;
public class test {}

Answer: B,E,G,H

Question No : 5

Given these facts about Java types in an application:

- Type x is a template for other types in the application.

- Type x implements dostuff ().

- Type x declares, but does NOT implement doit().

- Type y declares doOther() .

Which three are true?

A. Type y must be an interface.
B. Type x must be an abstract class.
C. Type y must be an abstract class.
D. Type x could implement or extend from Type y.
E. Type x could be an abstract class or an interface.
F. Type y could be an abstract class or an interface.

Answer: B,D,F

Explanation:

Unlike interfaces, abstract classes can contain fields that are not static and final, and they can containimplemented methods. Such abstract classes are similar to interfaces, except that they provide a partialimplementation, leaving it to subclasses to complete the implementation. If an abstract class contains onlyabstract method declarations, it should be declared as an interface instead.

Note:
An interface in the Java programming language is an abstract type that is used to specify an interface (in thegeneric sense of the term) that classes must implement. Interfaces are declaredusing the interface keyword,and may only contain method signature and constant declarations (variable declarations that are declared tobe both static and final). An interface maynever contain method definitions.

Note 2: an abstract class is a class that is declared abstract--it may or may not include abstract methods.Abstract classes cannot be instantiated, but they can be subclassed. An abstract method is a method that isdeclared without an implementation (without braces, and followed by a semicolon)

Question No : 6

Given:

```
public abstract class Account {
   abstract void deposit (double amt);
   public abstract Boolean withdraw (double amt);
}

public class CheckingAccount extends Account {

}
```

What two changes, made independently, will enable the code to compile?

A. Change the signature of Account to: public class Account.
B. Change the signature of CheckingAccount to: public abstract CheckingAccount
C. Implement private methods for deposit and withdraw in CheckingAccount.
D. Implement public methods for deposit and withdraw in CheckingAccount.
E. Change Signature of checkingAccount to: CheckingAccount implements Account.
F. Make Account an interface.

Answer: B,D

Explanation:

Compiler say:

- Der Typ CheckingAccount muss die übernommene abstrakte Methode Account.deposit(double) implementieren

- Der Typ CheckingAccount muss die übernommene abstrakte Methode Account.withdraw(double) implementieren

ODER

Typ CheckingAccount als abstract definieren

Question No : 7

Given:

```
interface Books {

    //insert code here

}
```

Which fragment, inserted in the Books interface, enables the code to compile?

A. public abstract String type;
public abstract String getType();
B. public static String type;
public abstract String getType();
C. public String type = "Fiction";
public static String getType();
D. public String type = "Fiction";
public abstract String getType();

Answer: D

Question No : 8

Given:

```
interface Event {
   String type = "Event";
   public void details();
}

class Quiz {
   static String type = "Quiz";
}

public class PracticeQuiz extends Quiz implements Event {
   public void details() {
      System.out.print(type);
   }

   public static void main(String[] args) {
      new PracticeQuiz().details();
      System.out.print(" " + type);
   }
}
```

What is the result?

A. Event Quiz

B. Event Event
C. Quiz Quiz
D. Quiz Event
E. Compilation fails

Answer: E

Question No : 9

Which two forms of abstraction can a programmer use in Java?

A. enums
B. interfaces
C. primitives
D. abstract classes
E. concrete classes
F. primitive wrappers

Answer: B,D

Explanation:

When To Use Interfaces

An interface allows somebody to start from scratch to implement your interface or implement your interface insome other code whose original or primary purpose was quite different from your interface. To them, yourinterface is only incidental, something that have to add on to thetheir code to be able to use your package. Thedisadvantage is every method in the interface must be public. You might not want to expose everything.

*When To Use Abstract classes

An abstract class, in contrast, provides more structure. It usually defines some default implementations andprovides some tools useful for a full implementation. The catch is, code using it must use your class as thebase. That may be highly inconvenient if the other programmers wanting to use your package have alreadydeveloped their own class hierarchy independently. In Java, a class can inherit from only one base class.*When to Use Both

You can offer the best of both worlds, an interface and an abstract class. Implementors can ignore yourabstract class if they choose. The only drawback of doing that is calling methods via their interface name isslightly slower than calling them via their abstract class name.

Reference:http://mindprod.com/jgloss/interfacevsabstract.html