



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:

Static method 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 various objects and then the super.equals() is invoked, the object method equals() what still result in FALSE better 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 contain implemented methods. Such abstract classes are similar to interfaces, except that they provide a partial implementation, leaving it to subclasses to complete the implementation. If an abstract class contains only abstract 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 the generic sense of the term) that classes must implement. Interfaces are declared using the interface keyword, and may only contain method signature and constant declarations (variable declarations that are declared to be both static and final). An interface may never 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 is declared 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 in some other code whose original or primary purpose was quite different from your interface. To them, your interface is only incidental, something that has to be added on to their code to be able to use your package. The disadvantage 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 and provides some tools useful for a full implementation. The catch is, code using it must use your class as the base. That may be highly inconvenient if the other programmers wanting to use your package have already developed 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 your abstract class if they choose. The only drawback of doing that is calling methods via their interface name is slightly slower than calling them via their abstract class name.

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

Question No : 10

Given:

```
class Plant {
    abstract String growthDirection();
}

class Embryophyta extends Plant {
    String growthDirection() { return "Up " }
}

public class Garden {
    public static void main(String[] args) {
        Embryophyta e = new Embryophyta();
        Embryophyta c = new Carrot();
        System.out.print(e.growthDirection() + growthDirection());
    }
}
```

What is the result?

- A. Up Down
- B. Up Up
- C. Up null
- D. Compilation fails
- E. An exception is thrown at runtime

Answer: D

Explanation:

Exception in thread "main" java.lang.ExceptionInInitializerError at garden.Garden.main
Caused by: java.lang.RuntimeException: Uncompilable source code - garden.Plant is not abstract and doesnot override abstract method growthDirection() in garden.Plant

Question No : 11

Given the classes:

```
class Pupil {  
    String name = "unknown";  
    public String getName() { return name; }  
}  
  
class John extends Pupil {  
    String name = "John";  
}  
  
class Harry extends Pupil {  
    String name = "Harry";  
    public String getName() { return name; }  
}  
  
public class Director {  
    public static void main(String[] args) {  
        Pupil p1 = new John();  
        Pupil p2 = new Harry();  
        System.out.print(p1.getName() + " ");  
        System.out.print(p2.getName());  
    }  
}
```

What is the result?

- A. John Harry
- B. unknown Harry
- C. john unknown
- D. unknown unknown
- E. Compilation fails.
- F. An exception is thrown at runtime.

Answer: B

Explanation:

getName() is missing in John, hence Pupils getName() is invoked and the String in Pupils scope returned.

Question No : 12

Given:

```
interface Writable {  
    void write (String s);  
}  
  
abstract class Writer implements Writable {  
    // Line ***  
}
```

Which two statements are true about the writer class?

- A. It compiles without any changes.
- B. It compiles if the code void write (String s); is added at line***.
- C. It compiles if the code void write (); is added at line ***.
- D. It compiles if the code void write (string s) { } is added at line ***.
- E. It compiles if the code write () {}is added at line ***.

Answer: A

Explanation:

An abstract class does not need to implement the interface methods.

Question No : 13

Given the two Java classes:

```
class Word {  
  
    private Word(int length) {}  
    protected Word(String w) {}  
  
}  
  
public class Buzzword extends Word {  
    public Buzzword() {  
        // Line ***, add code here  
    }  
  
    public Buzzword(String s) {  
        super(s);  
    }  
}
```

Which two code snippets, added independently at line ***, can make the Buzzword class compile?

- A. this ();
- B. this (100);
- C. this ("Buzzword");
- D. super ();
- E. super (100);
- F. super ("Buzzword");

Answer: C,F

Question No : 14

Given:

```
public enum Direction {  
  
    NORTH, EAST, SOUTH, WEST  
  
}
```

Which statement will iterate through Direction?

- A.

```
for (Direction d : Direction.values()){  
//  
}
```
- B.

```
for (Direction d : Direction.asList()){  
//  
}
```
- C.

```
for (Direction d : Direction.iterator()){  
//  
}
```
- D.

```
for (Direction d : Direction.asArray()){  
//  
}
```

Answer: A

Explanation:

The static values() method of an enum type returns an array of the enum values. The foreach loop is a good way to go over all of them.

//... Loop over all values.

```
for (Direction d : Direction.values()){  
System.out.println(d); // PrintsNORTH, EAST, ...  
}
```

Question No : 15

Given:

```
public class Runner {  
    public static String name = "unknown";  
    public void start() {  
        System.out.println(name);  
    }  
    public static void main(String[] args) {  
        name = "Daniel";  
        start();  
    }  
}
```

What is the result?

- A. Daniel
- B. Unknown
- C. It may print "unknown" or "Daniel" depending on the JVM implementation.
- D. Compilation fails.
- E. An exception is thrown at runtime.

Answer: D

Explanation:

The compilation fails at line `start();`

Erstellen eines statischen Verweises auf die nicht statische Methode `start()` vom Typ `Runner` nicht möglich. Exception in thread "main" `java.lang.RuntimeException:`

Uncompilable source code - non-static method `start()` cannot be referenced from a static context

Question No : 16

Which four are true about enums?

- A. An enum is typesafe.
- B. An enum cannot have public methods or fields.
- C. An enum can declare a private constructor.
- D. All enums implicitly implement `Comparable`.
- E. An enum can subclass another enum.
- F. An enum can implement an interface.

Answer: A,C,D,F

Explanation:

C: The constructor for an enum type must be package-private or private access.

Reference: Java Tutorials, Enum Types

Question No : 17

Given:

```
public class Task {
    String title;
    static class Counter {
        int counter = 0;
        void increment() { counter++; }
    }

    public static void main(String[] args) {
        // insert code here

    }
}
```

Which statement, inserted at line 8, enables the code to compile?

- A. new Task().new Counter().increment();
- B. new Task().Counter().increment();
- C. new Task.Counter().increment();
- D. Task.Counter().increment();
- E. Task.Counter.increment();

Answer: C

Question No : 18

Which represents part of a DAO design pattern?

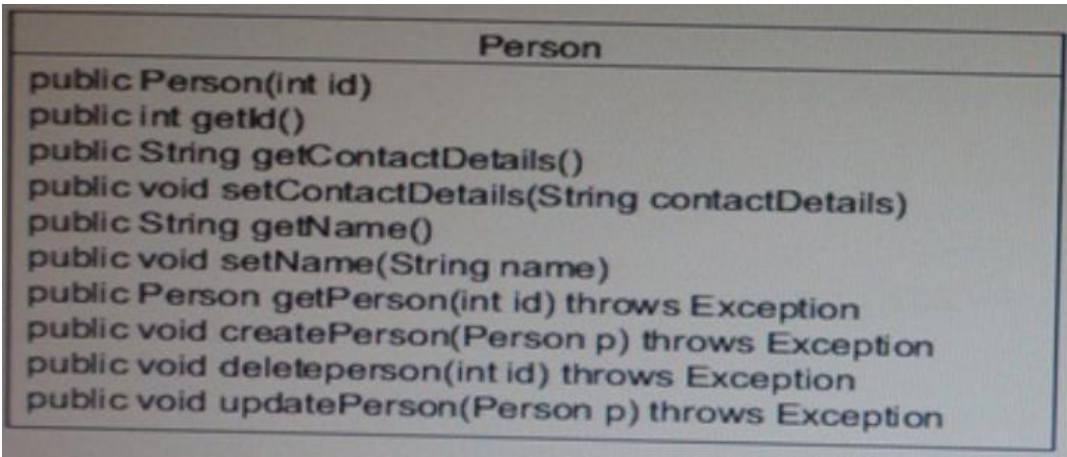
- A. interface EmployeeDAO {
int getID();
Employee findByID (intid);
void update();
void delete();
}
- B. class EmployeeDAO {
int getID() { return 0;}
Employee findByID (int id) { return null;}
void update () {}

```
void delete () {}  
}  
C. class EmployeeDAO {  
void create (Employee e) {}  
void update (Employee e) {}  
void delete (int id) {}  
Employee findById (int id) {return id}  
}  
D. interface EmployeeDAO {  
void create (Employee e);  
void update (Employee e);  
void delete (int id);  
Employee findById (int id);  
}  
E. interface EmployeeDAO {  
void create (Connection c, Employee e);  
void update (Connection c, Employee e);  
void delete (Connection c, int id);  
Employee findById (Connection c, int id);  
}
```

Answer: D

Question No : 19

Given:



```
Person  
public Person(int id)  
public int getId()  
public String getContactDetails()  
public void setContactDetails(String contactDetails)  
public String getName()  
public void setName(String name)  
public Person getPerson(int id) throws Exception  
public void createPerson(Person p) throws Exception  
public void deletePerson(int id) throws Exception  
public void updatePerson(Person p) throws Exception
```

Which group of method is moved to a new class when implementing the DAO pattern?

A. public in getId ()

```
public String getContractDetails ()
public Void setContractDetails(String contactDetails)
public String getName ()
public void setName (String name)
B. public int getId ()
public String getContractDetails()
public String getName()
public Person getPerson(int id) throws Exception
C. public void setContractDetails(String contractDetails) public void setName(String name)
D. public Person getPerson(int id) throws Exception
public void createPerson(Person p) throws Exception
public void deletePerson(int id) throws Exception
public void updatePerson(Person p) throws Exception
```

Answer: D

Explanation:

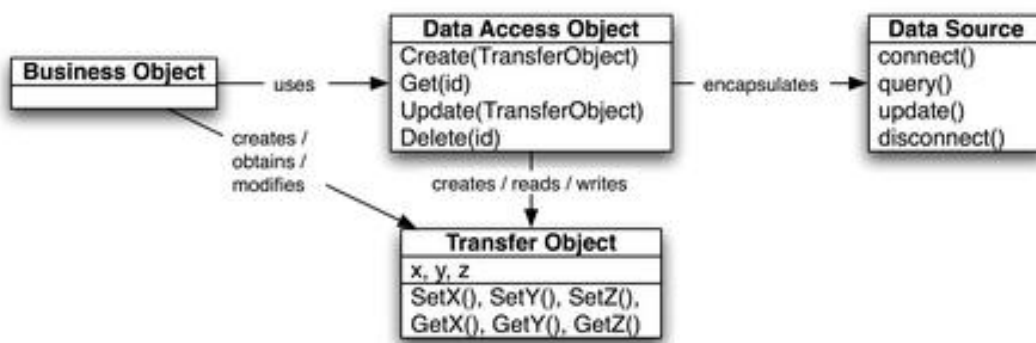
The methods related directly to the entity Person is moved to a new class.

CRUD

Note:DAO Design Pattern

*Abstracts and encapsulates all access to a data source *Manages the connection to the data source to obtain

and store data *Makes the code independent of the data sources and data vendors (e.g. plain-text, xml, LDAP, MySQL, Oracle, DB2)



D:\Documents and Settings\useralbo\Desktop\1.jpg

Example (here Customer is the main entity):

```
public class Customer {
private final String id;
private String contactName;
private String phone;
public void setId(String id) { this.id = id; }
public String getId() { return this.id; }
```

```
public void setContactName(String cn) { this.contactName = cn;} public String
getContactName() { return
this.contactName; } public void setPhone(String phone) { this.phone = phone; } public
String getPhone()
{ return this.phone; }
}
public interface CustomerDAO {
public void addCustomer(Customer c) throws DataAccessException; public Customer
getCustomer(String id)
throws DataAccessException; public List getCustomers() throws DataAccessException;
public void
removeCustomer(String id) throws DataAccessException; public void
modifyCustomer(Customer c) throws
DataAccessException; }
57
```

Question No : 20

Given:

```
interface Car {
    public void start();
}
class BasicCar implements Car {
    public void start() {}
}
public class SuperCar {
    Car c = new BasicCar ();
    public void start() {
        c.start();
    }
}
```

Which three are true?

- A. BasicCar uses composition.
- B. SuperCar uses composition.
- C. BasicCar is-a Car.
- D. SuperCar is-a Car.
- E. SuperCar takes advantage of polymorphism
- F. BasicCar has-a Car

Answer: B,C,E

Explanation:

B: The relationship modeled by composition is often referred to as the "has-a" relationship. Here SuperCar has-a Car.

C: The relationship modeled by inheritance is often referred to as the "is-a" relationship. Modeling an is-a relationship is called inheritance because the subclass inherits the interface and, by default, the implementation of the superclass. Inheritance of interface guarantees that a subclass can accept all the same messages as its superclass. A subclass object can, in fact, be used anywhere a superclass object is called for. E: The polymorphic method call allows one type to express its distinction from another, similar type, as long as they're both derived from the same base type. This distinction is expressed through differences in behavior of the methods that you can call through the base class.

Question No : 21

Given:

```
import java.util.ArrayList;
import java.util.List;

interface Glommer { }
interface Plinkable { }

public class Flimmer implements Plinkable {
    List<Tagget> t = new ArrayList<Tagget>();
}

class Flommer extends Flimmer { }

class Tagget {
    void doStuff() {
        String s = "yo";
    }
}
```

Which three statements concerning the OO concepts "is-a" and "has-a" are true?

- A. Flimmer is-a Plinkable
- B. Flommer has-a Tagget
- C. Flommer is-a Glommer
- D. Tagget has-a String
- E. Flommer is-a Plinkable
- F. Flimmer is-a Flommer
- G. Tagget is-a Plinkable

Answer: A,B,E

Explanation:

A: Flimmer implements Plinkable.

Flimmer is-a plinkable.

D:The relationship modeled by composition is often referred to as the "has-a" relationship.
Here Tagget has a String.

F: Flommer extends Flimmer

So there is an "is-a" relationship between Flommer and Flimmer .

Note: The has-a relationship has an encapsulation feature (like private or protected modifier used before each member field or method).

Question No : 22

Which two compile?

- A. interface Compilable {
void compile();
}
- B. interface Compilable {
final void compile();
}
- C. interface Compilable {
static void compile();
}
- D. interface Compilable {
abstract void compile();
}
- E. interface Compilable {
protected abstract void compile ();
}

Answer: A,D

Question No : 23

Which is a key aspect of composition?

- A. Using inheritance
- B. Method delegation
- C. Creating abstract classes
- D. Implementing the composite interface

Answer: B

Explanation:

In the composition approach, the subclass becomes the "front-end class," and the superclass becomes the "back-end class." With inheritance, a subclass automatically inherits an implementation of any non-private superclass method that it doesn't override. With composition, by contrast, the front-end class must explicitly invoke a corresponding method in the back-end class from its own implementation of the method. This explicit call is

sometimes called "forwarding" or "delegating" the method invocation to the back-end object. Note: Composition means the same as:

* contains

* is part of

Note 2: As you progress in an object-oriented design, you will likely encounter objects in the problem domain that contain other objects. In this situation you will be drawn to modeling a similar arrangement in the design of your solution. In an object-oriented design of a Java program, the way in which you model objects that contain other objects is with composition, the act of composing a class out of references to other objects.

With composition, references to the constituent objects become fields of the containing object. To use composition in Java, you use instance variables of one object to hold references to other objects.

Question No : 24

Given:

```
public class Customer {
    private int id;
    private String name;

    public int getId() { }
    public String getName() { }
    public boolean add(Customer new) { }
    public void delete(int id) { }
    public Customer find(int id) { }
    public boolean update(Customer cust) { }
}
```

What two changes should you make to apply the DAO pattern to this class?

- A. Make the Customer class abstract.
- B. Make the customer class an interface.
- C. Move the add, delete, find, and update methods into their own implementation class.
- D. Create an interface that defines the signatures of the add, delete, find, and update methods.
- E. Make the add, delete, and find, and update methods private for encapsulation.
- F. Make the getName and getID methods private for encapsulation.

Answer: C,D

Explanation:

C: The methods related directly to the entity Customer is moved to a new class.

D: Example (here Customer is the main entity):

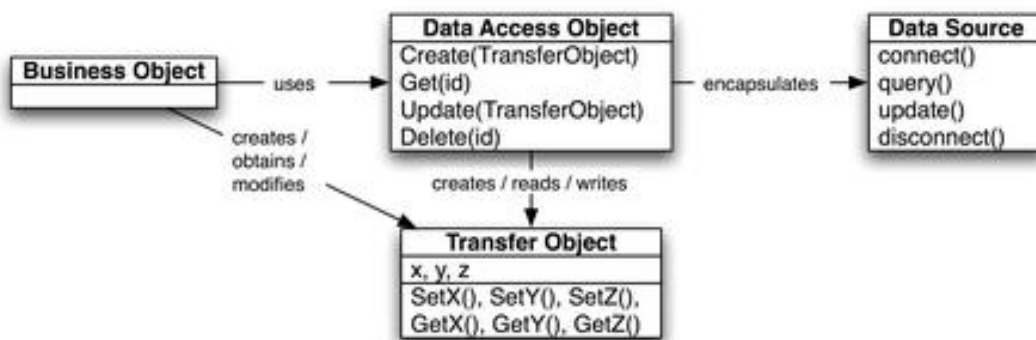
```
public class Customer {
private final String id;
private String contactName;
private String phone;
public void setId(String id) { this.id = id; }
102
```

```
public String getId() { return this.id; }
public void setContactName(String cn) { this.contactName = cn;} public String
getContactName() { return
this.contactName; } public void setPhone(String phone) { this.phone = phone; } public
String getPhone()
{ return this.phone; }
}
```

```
public interface CustomerDAO {
public void addCustomer(Customer c) throws DataAccessException; public Customer
getCustomer(String id) throws DataAccessException; public List getCustomers() throws
DataAccessException; public void
removeCustomer(String id) throws DataAccessException; public void
modifyCustomer(Customer c) throws
DataAccessException; }
```

Note: DAO Design Pattern

*Abstracts and encapsulates all access to a data source *Manages the connection to the data source to obtain and store data *Makes the code independent of the data sources and data vendors (e.g. plain-text, xml, LDAP, MySQL, Oracle, DB2)



D:\Documents and Settings\useralbo\Desktop\1.jpg

Question No : 25

Which two are true about Singletons?

- A. A Singleton must implement serializable.
- B. A Singleton has only the default constructor.
- C. A Singleton implements a factory method.
- D. A Singleton improves a class's cohesion.
- E. Singletons can be designed to be thread-safe.

Answer: C,E

Question No : 26

What are two differences between Callable and Runnable?

- A. A Callable can return a value when executing, but a Runnable cannot.
- B. A Callable can be executed by a ExecutorService, but a Runnable cannot.
- C. A Callable can be passed to a Thread, but a Runnable cannot.
- D. A Callable can throw an Exception when executing, but a Runnable cannot.

Answer: A,D

Explanation:

The Callable interface is similar to Runnable, in that both are designed for classes whose instances are potentially executed by another thread. A Runnable, however, does not return a result and cannot throw a checked exception.

Question No : 27

Which two properly implement a Singleton pattern?

- A. class Singleton {

```
private static Singleton instance;  
private Singleton () {}  
public static synchronized Singleton getInstance() {  
if (instance == null) {  
instance = new Singleton ();  
}  
return instance;  
}  
}
```

```
B. class Singleton {  
private static Singleton instance = new Singleton();  
protected Singleton () {}  
public static Singleton getInstance () {  
return instance;  
}  
}
```

```
C. class Singleton {  
Singleton () {}  
private static class SingletonHolder {  
private static final Singleton INSTANCE = new Singleton ();  
}  
public static Singleton getInstance () {  
return SingletonHolder.INSTANCE;  
}  
}
```

```
D. enum Singleton {  
INSTANCE;  
}
```

Answer: A,D

Explanation:

A: Here the method for getting the reference to the Singleton object is correct.

B: The constructor should be private

C: The constructor should be private

Note: Java has several design patterns Singleton Pattern being the most commonly used. Java Singleton pattern belongs to the family of design patterns, that govern the instantiation process. This design pattern proposes that at any time there can only be one instance of a singleton (object) created by the JVM.

The class's default constructor is made private, which prevents the direct instantiation of the object by others (Other Classes). A static modifier is applied to the instance method that returns the object as it then makes this method a class level method that can be accessed without creating an object.

OPTION A == SHOW THE LAZY initialization WITHOUT DOUBLE CHECKED LOCKING

TECHNIQUE ,BUT

ITS CORRECT

OPTION D == Serialzation and thraead-safety guaranteed and with couple of line of code
enum Singletonpattern is best way to create Singleton in Java 5 world.

AND THERE ARE 5 WAY TO CREATE SINGLETON CLASS IN JAVA

1>>LAZY LOADING (initialization) USING SYCHRONIZATION

2>>CLASS LOADING (initialization) USINGprivate static final Singleton instance = new
Singleton();

3>>USING ENUM

4>>USING STATIC NESTED CLASS

5>>USING STATIC BLOCK

AND MAKE CONSTRUCTOR PRIVATE IN ALL 5 WAY.

Question No : 28

Given:

```
import java.util.ArrayList;
import java.util.List;

interface Glommer { }
interface Plinkable { }

public class Flimmer implements Plinkable {
    List<Tagget> t = new ArrayList<Tagget>();
}

class Flommer extends Flimmer {
    String s = "hey";
}

class Tagget implements Glommer {
    void doStuff() {
        String s = "yo";
    }
}
```

Which two statements concerning the OO concepts "IS-A" and "HAS-A" are true?

- A. Flimmer is-a Glommer.
- B. Flommer has-a String.
- C. Tagget has-a Glommer.
- D. Flimmer is-a ArrayList.
- E. Tagget has-a doStuff()
- F. Tagget is-a Glommer.

Answer: B,F

Explanation:

B: The relationship modeled by composition is often referred to as the "has-a" relationship. Here Flommer has-a String.

E: The has-a relationship has an encapsulation feature (like private or protected modifier used before each member field or method).

Here Tagget has-a method doStuff()

F: Tagget implements Glommer.

Tagget is-a Glommer.

Note: The has-a relationship has an encapsulation feature (like private or protected modifier used before each member field or method).

Question No : 29

Given the integer implements comparable:

```
import java.util.*;

public class SortAndSearch2 {
    static final Comparator<Integer> IntegerComparator = new Comparator<Integer>() {
        public int compare (Integer n1, Integer n2) {
            return n2.compareTo(n1);
        }
    };

    public static void main(String args[]) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add (4);
        list.add (1);
        list.add (3);
        list.add (2);

        Collections.sort(list, null);
        System.out.println(Collections.binarySearch(list, 3));

        Collections.sort(list,IntegerComparator);
        System.out.println(Collections.binarySearch(list, 3));
    }
}
```

What is the result?

- A. 4
1
- B. 1
2
- C. 32
- D. 21
- E. 2
3

Answer: D

Explanation:

binarySearch

public static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key)

Searches the specified list for the specified object using the binary search algorithm.

The list must be sorted into ascending order according to the natural ordering of its elements (as by the sort(List) method) prior to making this call. If it is not sorted, the results are undefined.

Parameters:

list - the list to be searched.

key - the key to be searched for.

Returns:

the index of the search key, if it is contained in the list; otherwise, $-(\text{insertion point}) - 1$.

Question No : 30

Which statement declares a generic class?

- A. public class Example < T > { }
- B. public class <Example> { }
- C. public class Example <> { }
- D. public class Example (Generic) { }
- E. public class Example (G) { }
- F. public class Example { }

Answer: A

Explanation:

Example:

```
public class Pocket<T>
{
private T value;
public Pocket() {}
public Pocket( T value ) { this.value = value; }
public void set( T value ) { this.value = value; }
public T get() { return value; }
public boolean isEmpty() { return value != null; }
public void empty() { value = null; }
}
```

Question No : 31

Given:

```
Deque <String> myDeque = new ArrayDeque<String>();  
  
myDeque.push("one");  
myDeque.push("two");  
myDeque.push("three");  
  
System.out.println(myDeque.pop());
```

What is the result?

- A. Three
- B. One
- C. Compilation fails.
- D. The program runs, but prints no output.

Answer: A

Explanation:

push

void push(E e)

Pushes an element onto the stack represented by this deque (in other words, at the head of this deque) if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an `IllegalStateException` if no space is currently available.

This method is equivalent to `addFirst(E)`.

pop

E pop()

Pops an element from the stack represented by this deque. In other words, removes and returns the first element of this deque.

This method is equivalent to `removeFirst()`.

Returns:

the element at the front of this deque (which is the top of the stack represented by this deque)

Throws:

`NoSuchElementException` - if this deque is empty

Question No : 32

Given the following code fragment:

```
public static void main(String[] args) {  
    Connection conn = null;  
    Deque<String> myDeque = new ArrayDeque<>();  
    myDeque.add("one");  
    myDeque.add("two");  
    myDeque.add("three");  
    System.out.println(myDeque.remove());  
}
```

What is the result?

- A. Three
- B. One
- C. Compilation fails
- D. The program runs, but prints no output

Answer: B

Explanation:

add

boolean add(E e)

Inserts the specified element into the queue represented by this deque (in other words, at the tail of this deque) if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an `IllegalStateException` if no space is currently available. When using a capacity-restricted deque, it is generally preferable to use `offer`.

This method is equivalent to `addLast(E)`.

remove

E remove()

Retrieves and removes the head of the queue represented by this deque (in other words, the first element of this deque). This method differs from `poll` only in that it throws an exception if this deque is empty.

This method is equivalent to `removeFirst()`.

Returns:

The head of the queue represented by this deque

Class `ArrayDeque`

Question No : 33

Which concept allows generic collections to interoperate with java code that defines collections that use rawtypes?

- A. bytecode manipulation
- B. casting
- C. autoboxing
- D. auto-unboxing
- E. type erasure

Answer: E

Explanation:

The type erasure of its leftmost bound, or type Object if no bound was specified.

Examples:

type parameters type erasure

List<String> List

Map.Entry<String,Long> Map.Entry

<T extends Cloneable & Comparable<T>> Cloneable

<T extends Object & Comparable<T>> Object

<T> T[] toArray(T[] a) Object[] toArray(Object[] a)

The type erasure process can be imagined as a translation from generic Java source code back into regularJava code. In reality the compiler is more efficient and translates directly to Java byte code. But the byte codecreated is equivalent to the non-generic Java code.

Question No : 34

Given:

```
public class Test {  
    Integer x; // line 2  
  
    public static void main(String[] args) {  
        new Test().go(5);  
    }  
  
    void go(Integer i) { // line 6  
        System.out.print(x + ++i); // line 7  
    }  
}
```

What is the result?

- A. 5
- B. 6
- C. An exception is thrown at runtime
- D. Compilation fails due to an error on line 6
- E. Compilation fails due to an error on line 7

Answer: C

Explanation:

The code compile fine but java.lang.NullPointerException is thrown at runtime.

x has no value. The code would run if line 2 was changed to:

Integer x = 3;

Question No : 35

Given:

```
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;

public class MapClass {
public static void main(String[] args) {
    Map <String, String> partList = new TreeMap<>();
    partList.put("P002", "Large Widget");
    partList.put("P001", "Widget");
    partList.put("P002", "X-Large Widget");

    Set<String> keys = partList.keySet();

    for (String key:keys) {
        System.out.println(key + " " + partList.get(key));
    }
}
```

What is the result?

- A. p001 Widget
p002 X-Large Widget
- B. p002 Large Widget
p001 Widget
- C. p002 X-large Widget
p001 Widget
- D. p001 Widget
p002 Large Widget
- E. compilation fails

Answer: A

Explanation: Compiles fine. Output is:

P001 Widget

P002 X-Large Widget

Line: partList.put("P002", "X-Large Widget"); >> overwrites >> line:partList.put("P002", "Large Widget");

put

V put(K key, V value)

Associates the specified value with the specified key in this map (optional operation). If the map previously contained a mapping for the key, the old value is replaced by the specified

value. (A map *m* is said to contain a mapping for a key *k* if and only if *m.containsKey(k)* would return true.)

Parameters:

key - key with which the specified value is to be associated

value - value to be associated with the specified key

Returns the previous value associated with key, or null if there was no mapping for key. (A null return can also indicate that the map previously associated null with key, if the implementation supports null values.)

Question No : 36

Given:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class NameList {
    public static void main(String[] args) {
        List<String> nameList = new ArrayList<>(3);

        nameList.add("John Adams");
        nameList.add("George Washington");
        nameList.add("Thomas Jefferson");

        Collections.sort(nameList);
        for (String name : nameList) {
            System.out.println(name);
        }
    }
}
```

What is the result?

- A. John Adams
George Washington
Thomas Jefferson
- B. George Washington
John Adams
Thomas Jefferson
- C. Thomas Jefferson
John Adams
George Washington
- D. An exception is thrown at runtime
- E. Compilation fails

Answer: B

Explanation: The program compiles and runs fine.

At runtime the NameList is built and then sorted by natural Order (String >> alphabetically).

Question No : 37

Given:

```
import java.util.ArrayDeque;
import java.util.Deque;

public class Counter {
    public static void main(String[] args) {
        Deque<String> deq = new ArrayDeque<String>(2);

        deq.addFirst("one");
        deq.addFirst("two");
        deq.addFirst("three"); // Line 9

        System.out.print(deq.pollLast());
        System.out.print(deq.pollLast());
        System.out.print(deq.pollLast()); // Line 12
    }
}
```

What is the result?

- A. An exception is thrown at runtime on line 9.
- B. An exception is thrown at runtime on line 12
- C. onetwonull
- D. onetwothree
- E. twoonenull
- F. threetwoone

Answer: D

Explanation:

addFirst

void addFirst(E e)

Inserts the specified element at the front of this deque if it is possible to do so immediately without violating

capacity restrictions. When using a capacity-restricted deque, it is generally preferable to use method offerFirst

(E).

pollLast

E pollLast()

Retrieves and removes the last element of this deque, or returns null if this deque is empty.

Returns:

the tail of this deque, or null if this deque is empty

Question No : 38

Given the class?

```
import java.util.ArrayList;
import java.util.Collections;

public class Name implements Comparable<Name> {
    String first, last;
    public Name(String first, String last) {
        this.first = first;
        this.last = last;
    }

    @Override
    public int compareTo(Name n) {
        int cmpLast = last.compareTo(n.last);
        return cmpLast != 0 ? cmpLast : first.compareTo(n.first);
    }

    public String toString() {
        return first + " " + last;
    }

    public static void main(String[] args) {
        ArrayList<Name> list = new ArrayList<Name>();
        list.add (new Name("Joe", "Shmoe"));
        list.add (new Name("John", "Doe"));
        list.add (new Name("Jane", "Doe"));
        Collections.sort(list);
        for (Name n : list) {
            System.out.println(n);
        }
    }
}
```

What is the result?

- A.** Jane Doe
John Doe
Joe Shmoe
- B.** John Doe
Jane Doe
Joe Shmoe
- C.** Joe Shmoe
John Doe

Jane Doe
D. Joe Shmoe
Jane Doe
John Doe
E. Jane Doe
Joe Shmoe
John Doe
F. John Doe
Joe Shmoe
Jane Doe

Answer: A

Explanation: The list will be sorted alphabetically (Lastname / Firstname).

first sorted by Lastname

if Lastname equals, sorted by firstname

Output will be:

Jane Doe
John Doe
Joe Shmoe

Question No : 39

Given the cache class:

```
public class Cache <T> {  
    private T t;  
    public void setValue (T t) { this.t=t; }  
    public T getValue() { return t; }  
}
```

What is the result of the following?

```
Cache<> c = new Cache<Integer>(); // Line 1  
SetValue(100); // Line 2  
System.out.print(c.getValue().intValue() +1); // Line 3
```

- A. 101
- B. Compilation fails at line 1.

- C. Compilation fails at line 2.
- D. Compilation fails at line 3.

Answer: B

Explanation:

Compilation failure at line:1

Incorrect number of arguments for type Cache<T>; it cannot be parameterized with arguments <>illegal start of typetype cache.Cache does not take parameters.

Question No : 40

Given:

```
public class StringSplit01 {
    public static void main(String[] args) {
        String names = "John.-George.-Paul.-Ringo";
        String[] results = names.split("-.");
        for(String str:results) {
            System.out.println(str);
        }
    }
}
```

What is the result?

- A. John.-George.-Paul.-Ringo
- B. John
George
Paul
Ringo
- C. John -
George -
Paul -
Ringo -
- D. An exception is thrown at runtime
- E. Compilation fails

Answer: B

Explanation:

The split() method is used to split a string into an array of substrings, and returns the new array.

regex: - followed by two characters

Question No : 41

Given:

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

class Test {
    private static String REGEX = "\\Sto\\S|\\bo\\b";
    private static String INPUT = "Nice to see you,to,be fine.";
    private static String REPLACE = ",";

    public static void main(String[] args) {
        Pattern p = Pattern.compile(REGEX);
        Matcher m = p.matcher(INPUT);
        INPUT = m.replaceAll(REPLACE);
        System.out.println(INPUT);
    }
}
```

What is the result?

- A. Nice to see you,be fine
- B. Nice,see you,be fine
- C. Nice,see you, to, be fine
- D. Nice, see you, be fine
- E. Nice to see y, u, be fine

Answer: A

Explanation:

The text ",to," is replaced by the ","

Question No : 42

Give:

```
public class Test {  
  
    public static void main(String[] args) {  
        String svar= "sports cars";  
        svar.replace(svar,"convertibles");  
        System.out.printf("There are %3$s %2$s and %d trucks.",5,svar,2+7);  
    }  
}
```

What is the result?

- A. There are 27 sports cars and 5 trucks
- B. There are 27 convertibles and 5 trucks
- C. There are 9 sports cars and 5 trucks
- D. There are 9 convertibles and 5 trucks
- E. `IllegalFormatConversionException` is thrown at runtime

Answer: C

Explanation:

Strings are immutable, therefore no change at line: `svar.replace(svar,"convertibles");`

Format String Syntax:

`%[argument_index$][flags][width][.precision]conversion`

The optional `argument_index` is a decimal integer indicating the position of the argument in the argument list.

The first argument is referenced by "1\$", the second by "2\$", etc.

The optional flags is a set of characters that modify the output format. The set of valid flags depends on the conversion.

's', 'S' general

'd' integral The result is formatted as a decimal / integer

Question No : 43

Given the code fragment:

```
String s = "Java 7, Java 6";  
Pattern p = Pattern.compile("Java.+\\d");  
Matcher m = p.matcher(s);  
while (m.find()) {  
    System.out.println(m.group());  
}
```

What is the result?

- A. Java 7
- B. Java 6
- C. Java 7, Java 6
- D. Java 7
java 6
- E. Java

Answer: C

Explanation:

regex: Java / one or more anything !!! / ends with a digit
so it is the source string

Question No : 44

Given:

```
import java.util.Scanner;

public class Painting {

    public static void main(String[] args) {
        String input = "Pastel, *Enamel, Fresco, *Gouache";
        Scanner s = new Scanner(input);
        s.useDelimiter(",\\s*");
        while (s.hasNext()) {
            System.out.println(s.next());
        }
        s.close();
    }
}
```

What is the result?

- A.** Pastel
Enamel
Fresco
Gouache
- B.** Pastel
*Enamel
Fresco
*Gouache
- C.** Pastel
Enamel
Fresco
Gouache
- D.** Pastel
Enamel, Fresco
Gouache

Answer: B

Explanation:

regex explanation:

, = ,

\ = masks the following

\s = A whitespace character: [\t \n \x0B \f \r]

* = Greedy Quantifier: zero or more times

Delimiter: comma + zero or more whitespace characters

Question No : 45

Given:

```
public class Test {  
  
    public static void main(String[] args) {  
        String[] arr = {"SE", "ee", "ME"};  
        for(String var : arr) {  
            try {  
                switch(var) {  
                    case "SE":  
                        System.out.println("Standard Edition");  
                        break;  
                    case "EE":  
                        System.out.println("Enterprise Edition");  
                        break;  
                    default: assert false;  
                }  
            } catch (Exception e) {  
                System.out.println(e.getClass());  
            }  
        }  
    }  
}
```

And the commands:

```
javac Test.java  
java ea Test
```

And the commands:

```
javac Test.java
```

java ea Test

What is the result?

A. Compilation fails

B. Standard Edition

Enterprise Edition

Micro Edition

C. Standard Edition

class java.lang.AssertionError

Micro Edition

D. Standard Edition is printed and an Assertion Error is thrown

Answer: D

Explanation:

javac Test.java

will compile the program.

As for command line:

java ea Test

First the code will produce the output:

Standard Edition

See Note below.

The ea option will enable assertions. This will make the following line in the switch statement to be run:

default: assert false;

This will throw an assertion error. This error will be caught. An the class of the assertion error (classjava.lang.AssertionError) will be printed by the following line:

System.out.println(e.getClass());

Note:The java tool launches a Java application. It does this by starting a Java runtime environment, loading aspecified class, and invoking that class's main method. The method declaration must look like the following:

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

Paramater ea:

```
-enableassertions[:<package name>"..." | :<class name> ] -ea[:<package name>"..." | :<class name> ]
```

Enable assertions. Assertions are disabled by default. With no arguments, enableassertions or -ea enablesassertions.

Note 2:

An assertion is a statement in the Java™ programming language that enables you to test your assumptionsabout your program.

Each assertion contains a boolean expression that you believe will be true when the assertion executes. If it is not true, the system will throw an error.

```
public class AssertionError extends Error
```

Thrown to indicate that an assertion has failed.

Note 3:

The javac command compiles Java source code into Java bytecodes. You then use the Java interpreter - the

java command - to interpret the Java bytecodes.

Reference: java - the Java application launcher

Reference: java.lang.Class AssertionError

Question No : 46

Given:

```
class InvalidAgeException extends IllegalArgumentException { }

public class Tracker {

    void verify (int age) throws IllegalArgumentException {
        if (age < 12)
            throw new InvalidAgeException ();
        if (age >= 12 && age <= 60)
            System.out.print("General category");
        else
            System.out.print("Senior citizen category");
    }

    public static void main(String[] args) {
        int age = Integer.parseInt(args[1]);
        try {
            new Tracker().verify(age);
        }
        catch (Exception e) {
            System.out.print(e.getClass());
        }
    }
}
```