Vendor: Oracle

Exam Code:1Z0-804

Exam Name: Java SE 7 Programmer II

Version: Demo

QUESTION 1

Given:

```
interface Rideable {
public String ride() { return "riding "; }
}
class Horse implements Rideable {
public String ride() { return "cantering "; }
}
class Icelandic extends Horse implements Rideable {
public String ride() { return "tolting "; }
}
class Test {
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. riding riding tolting

B. riding riding cantering

C. tolting cantering tolting

D. tolting cantering cantering

E. Compilation fails.

F. An exception is thrown at runtime.

Correct Answer: E

The compilation fails at:

interface Rideable {

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

}

Error due to: interface methods cannot have body.

QUESTION 2

View the exhibit: (*Missing*)

Given the code fragment:

class Finder extends SimpleFileVisitor {

private final PathMatcher matcher;

private static int numMatches = 0;

Finder () {

matcher = FileSystems.getDefault().getPathMatcher("glob:*java");

}

void find(Path file) {

Path name = file.getFileName();

if (name != null andand matcher.matches(name)) {

numMatches++;

}

}

void report()

{

System.out.println("Matched: " + numMatches);

}

@Override

public FileVisitResult visitFile(Path file, BasicFileAttributes attrs)

find(file);

return CONTINUE;

}

```
}

public class Visitor {

public static void main(String[] args) throws IOException {

Finder finder = new Finder();

Files.walkFileTree(Paths.get("D:\\Project"), finder);

finder.report();

}

}
```

What is the result?

A. Compilation fails

B. 6

C. 4

D. 1

E. 3

F. Not possible to answer due to missing exhibit.

Correct Answer: F

Note: The FileSystems.getDefault() returns the default FileSystem. The default file system creates objects that provide access to the file systems accessible to the Java virtual machine. The working directory of the file system is the current user directory, named by the system property user.dir.

QUESTION 3

Given the code fragment:

```
try {

String query = "SELECT * FROM Employee WHERE ID=110";

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(query);

System.out.println("Employee ID: " + rs.getInt("ID"));

} catch (Exception se) {

System.out.println("Error");

}
```

Assume that the SQL query matches one record. What is the result of compiling and executing this code?

A. The code prints Error.

B. The code prints the employee ID.

C. Compilation fails due to an error at line 13.

D. Compilation fails due to an error at line 14.

Correct Answer: B

The code compiles fine. The code will rune fine.

public int getInt(String columnName) throws SQLException Retrieves the value of the designated column in the current row of this ResultSet object as an int in the Java programming language

---

QUESTION 4

Which code fragment is required to load a JDBC 3.0 driver?

A. DriverManager.loadDr iver ("org.xyzdata.jdbc.NetworkDriver");

B. Class.forName("org.xyzdata.jdbc-NetworkDriver");

C. Connection con = Connection.getDriver ("jdbc:xyzdata://localhost:3306/EmployeeDB");

D. Connection con = DriverManager.getConnection ("jdbc:xyzdata://localhost:3306/EmployeeDB");

Correct Answer: B

In previous versions (prior to 4.0) of JDBC, to obtain a connection, you first had to initialize your JDBC driver by calling the method Class.forName. This methods required an object of type java.sql.Driver.

Note:

DriverManager: This fully implemented class connects an application to a data source, which is specified by a database URL. When this class first attempts to establish a connection, it automatically loads any JDBC 4.0 drivers found within

the class path. Note that your application must manually load any JDBC drivers prior to version 4.0.

QUESTION 5

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);

A. useDelimiter(",\\s*"); while (s.hasNext()) { System.out.println(s.next()); } } } What is the result?

B. Paste1 Ename1 Fresco Gouache

C. Paste1 *Ename1 Fresco *Gouache

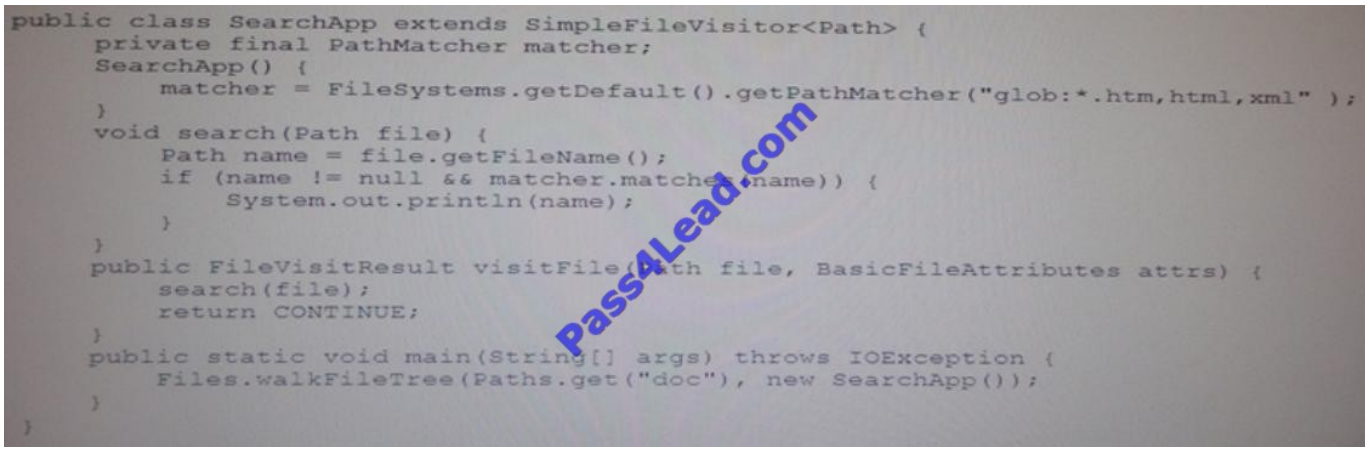
D. Pastel Ename1 Fresco Gouache

E. Pastel Ename1, Fresco Gouache

Correct Answer: B

QUESTION 6

Given the following files in doc directory: And the code fragment:



```
public class SearchApp extends SimpleFileVisitor<Path> {
      private final PathMatcher matcher;
      SearchApp() {
           matcher = FileSystems.getDefault().getPathMatcher("glob:*.htm,html,xml" );
      }
      void search(Path file) {
           Path name = file.getFileName();
           if (name != null && matcher.matches(name)) {
                System.out.println(name);
           }
      }
      public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) {
           search(file);
           return CONTINUE;
      }
      public static void main(String[] args) throws IOException {
           Files.walkFileTree(Paths.get("doc"), new SearchApp());
      }
}
```

What is the result, if doc is present in the current directory?

A. No output is produced.

B. index.htm

C. index.htm userguide.txt logo.gif

D. index.htm service.html userguide.txt logo.gif

Correct Answer: B

The Glob search expression is defined through "glob:*.htm, html, xml" Only the file name index.htm matches this pattern.

QUESTION 7

Which represents part of a DAO design pattern?

A. interface EmployeeDAO { int getID(); Employee findByID (int id); 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} }

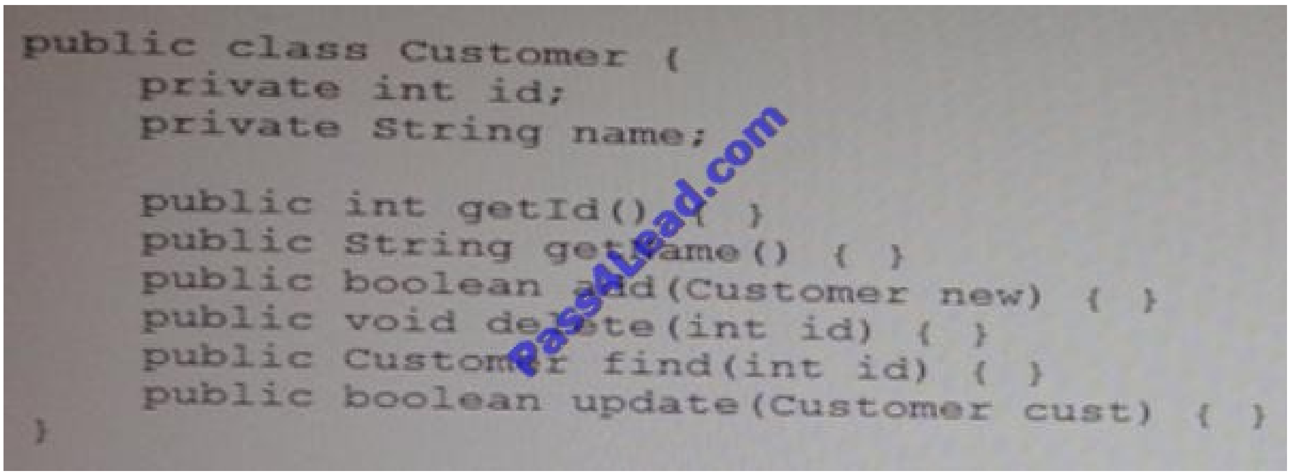
D. interface EmployeeDAO { void create (Employee e); void update (Employee e); void delete (int id); Employee findByID (int id); }

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); }

Correct Answer: D

QUESTION 8

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.

Correct Answer: CD

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; }

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

*

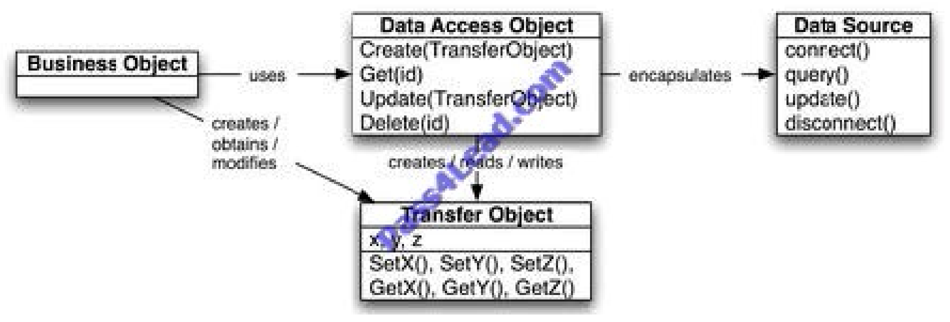
 Abstracts and encapsulates all access to a data source

*

 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)



QUESTION 9

Given:

class Car implements TurboVehicle, Steerable {

// Car methods > interface Convertible

{

// Convertible methods

}

public class SportsCar extends Car implements Convertible {

}

Which statement is true?

A. SportsCar must implement methods from TurboVehicle and steerable

B. SportsCar must override methods defined by car.

C. SportsCar must implement methods define by convertible.

D. Instances of car can invoke convertible methods.

Correct Answer: C

To declare a class that implements an interface, you include an implements clause in the class declaration.

By convention, theimplements clause follows the extends clause, if there is one.

Here are the some point that must be considered while implementing an interface (or interfaces) into a java class.

A class implementing an interface must either implement all the methods of that interface otherwise known as the abstract class.

A class in java may extend at most one superclass because java does not allow multiple inheritance, by it may implement more than one interface.

Multiple inheritance in java is achieved through the interfaces.

When a class implements more than one interface then implement statement requires a comma-separated list of interfaces to be implement by that class.

QUESTION 10

Given:

```
public class Test {

void display(String[] arr) {

try {

System.out.print(arr[2]);

} catch(ArrayIndexOutOfBoundsException |

NullPointerException e) {

e = new Exception();

throw e;

}

}

public static void main(String[] args) throws Exception {

try {

String[] arr = {"Unix","Solaris",null};

new Test().display(arr);

} catch(Exception e) {

System.err.print(e.getClass());

}

}

}
```

What is the result?

A. Null

B. class java.lang.ArraylndexOutOfBoundsException

C. class java.lang.NullPointerException

D. class java.lang.Exception

E. Compilation fails.

Correct Answer: E

error: incompatible types e = new Exception(); required: RuntimeException found: Exception

---

QUESTION 11

Given the incomplete pseudo-code for a fork/join framework application: submit(Data) { if(Data.size

A. Insert submit at line X.

B. Insert splitInHalf at line X.

C. Insert process at line X.

D. Insert process at line Y.

E. Insert splitInHalf at line Y.

F. Insert process at line Z.

G. Insert submit at line Z.

Correct Answer: CEG

C: If data is small enough then process it. Line X

E: If data is not small enough then split it half. Line Y

G: After the data has been split (line Y) then recursively submit the splitted data (Line z).

QUESTION 12

Given:

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

Set 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

Correct Answer: A

Compiles fine. Output is: P001 Widget P002 X-Large Widget Line: partList.put("P002", "X-Large Widget"); overwrites line: partList.put("P002", "Large Widget");