Correction Java Class Design - 0

C - 1

This is an example of Singleton class. static fields are initialize once, when class loads in the memory.

‘printer’ is static reference variable defined in PrinterCreator (static nested) class.

Printer p1 = Printer.getInstance(); => getInstance method loads the PrinterCreator class in the memory causing ‘static Printer printer = new Printer();’ to get executed. object of Printer class is created and variable count is incremented by 1.

Later invocation of Printer.getInstance(); method simply returns the reference of Printer object so Printer class’s constructor is not invoked.

Variable count is incremented only once.

Correction Java Class Design - 1

C - Compilation error

getClass(), notify(), notifyAll() and overloaded wait methods are final in Object class and hence cannot be overridden.

Correction Java Class Design - 2

B - 9999

You can write statements inside initialization blocks, variable a is of static type so both static and instance initialization blocks can access it.

Instance of Initializer block is not created in this case, so instance initialization block is not executed.

Execution of static initialization block decrements the value of a by 1. Hence the output is 9999.

Correction Java Class Design - 3

A - Code compiles successfully and on execution always prints “com.training.ocp.Player@64” on to the console

If toString() method is not overridden, then Object class’s version is invoked.

The toString() method in Object class has below definition:

public String toString() { return getClass().getName() + “@” + Integer.toHexString(hashCode()); }

So, in the output you get: fully-qualified-name-of-the-class@hexadecimal-representation-of-hash-code. NOTE: hashCode() method is called for that.

Player class overrides the hashCode() method so, toString() method calls this overriding hashCode() method. Output string will always contain 64.

Correction Java Class Design - 4

A - No

Immutable class should have private fields and no setters.

In this case it is possible to change the msg after an instance of Greet class is created. Check below code:

```
public class Test {
    public static void main(String[] args) {
        Greet greet = new Greet("Hello"); //msg refers to "Hello"
        greet.setMsg("Welcome"); //msg refers to "Welcome"
        System.out.println(greet.getMsg()); //Prints "Welcome" on to
         the console.
    }
}
```

To make Greet class immutable, delete the setter and add modifier ‘final’ for variable msg.

```
final class Greet {
    private final String msg;
    public Greet(String msg) {
        this.msg = msg;
    }

    public String getMsg() {
        return msg;
    }
}
```

Once value is assigned to msg variable in the constructor, it cannot be changed later.

Correction Java Class Design - 5

C - Parent

Parent is a concrete class.

In java it is possible for abstract class to not have any abstract methods.

Class Child is abstract, it extends Parent and doesn't contain any abstract method. It contains special main method, so JVM can invoke this method.

Rest of the code is normal java code. new Parent().m(); creates an instance of Parent class and invokes method m() on that instance. "Parent" is printed on to the console.

Correction Java Class Design - 6

A - false

Object class contains:

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

Class Player doesn't override equals(Object) method rather it overloads the equals method: equals(Player).

As p2 is of Object type, p1.equals(p2) invokes the equals(Object) method defined in Object class and as this method compares the reference (memory address), hence it returns false in this case.

Below code will correctly override equals(Object) method in the Player class:

```
public boolean equals(Object player) {
    if(player instanceof Player) {
        Player p = (Player)player;
        if(this.name.equals(p.name) && this.age == p.age)
            return true;
    }
    return false;
}
```

Correction Java Class Design - 7

D - Compilation error in Point class

To call another constructor of the class use this(10, 20);

No-argument constructor of Point class causes compilation error.

Correction Java Class Design - 8

B - Vaccinating…

Abstract class can have static methods and those can be called by using Class_Name.method_name.

class Animal is declared with package(default) scope, which means it can be extended by other classes in the same package (com.training.ocp).

Above code executes fine and prints ‘Vaccinating…’ on to the console.

Correction Java Class Design - 9

A - Compilation error

The toString() method in Object class has below definition:

```
public String toString() {
    return getClass().getName() + "@" +
        Integer.toHexString(hashCode());
}
```

class Player doesn't override it correctly, return type should be String and not Object.

Correction Java Class Design - 10

C - None of the other options

If toString() method is not overridden, then Object class's version is invoked.

The toString() method in Object class has below definition:

```
public String toString() {
    return getClass().getName() + "@" +
        Integer.toHexString(hashCode());
}
```

So, in the output you get: fully-qualified-name-of-the-class@hexadecimal-representation-of-hash-code. NOTE: hashCode() method is called for that.

Player class doesn't override the hashCode() method, rather it defines a new method hashcode() [NOTE: c in lower case in the method name].

Hence, hashcode() is never invoked and no guarantee of getting 64 always.

Correction Java Class Design - 11

D - Name: null, Age: 0

Methods can have same name as the class. Player() is method and not constructor of the class, note the void return type of this method.

As no constructors are provided in the Player class, java compiler adds default no-argument constructor. That is why “new Player()” doesn’t cause any compilation error.

Default values are assigned to instance variables, hence null is assigned to name and 0 is assigned to age.

In the output, Name: null, Age: 0 is displayed.

Correction Java Class Design - 12

A - The sum is: 1525

Operator + is left to right associative, so given expression can be grouped as:

“The sum is:” + x + y

= (“The sum is:” + x) + y

= ((“The sum is:” + x) + y)

= ((“The sum is:” + 15) + 25)

= (“The sum is: 15” + 25)

= “The sum is: 1525”

Correction Java Class Design - 13

A - Runtime exception

Dog and Cat are siblings as both extend from Animal class.

animals refer to an instance of Dog [] type. Each element of Dog [] can store Dog instances and not Cat instances.

But as we are using reference variable of Animal [] type hence compiler allows to add both Cat and Dog instances.

So, animals[0] = new Dog(); and animals[1] = new Cat(); don’t cause any compilation error.

But at runtime, while executing the statement: “animals[1] = new Cat();”, a Cat instance is assigned to Dog array’s element hence java.lang.ArrayStoreException is thrown.

Correction Java Class Design - 14

D - Compilation error

The toString() method in Object class has below definition:

```
public String toString() {
    return getClass().getName() + "@" +
        Integer.toHexString(hashCode());
}
```

toString() method cannot be overridden using protected modifier.

Correction Java Class Design - 15

D - Runtime exception

animals refer to an instance of Dog [] type. Each element of Dog [] can store Dog instances but not Animal instances(sub type can’t refer to super type).

But as we are using reference variable of Animal [] type hence compiler allows to add both Animal and Dog instances.

So, animals[0] = new Animal(); and animals[1] = new Dog(); don’t cause any compilation error.

But at runtime, while executing the statement: “animals[0] = new Animal();”, an Animal instance is assigned to Dog array’s element hence java.lang.ArrayStoreException is thrown.

Correction Java Class Design - 16

B - Compilation error in Animal class

‘abstract’ and ‘private’ cannot be used together.