

# Review Test Submission: Quiz 11 - OODP

|                   |   |
|-------------------|---|
| User              | Kaidi He  |
| Course            | 201503_Advanced Software Paradigms_CSCI_6221_11 |
| Test              | Quiz 11 - OODP                                  |
| Started           | 11/18/15 6:12 PM                                |
| Submitted         | 11/18/15 6:46 PM                                |
| Due Date          | 11/18/15 6:50 PM                                |
| Status            | Completed                                       |
| Attempt Score     | 70 out of 100 points                            |
| Time Elapsed      | 33 minutes out of 40 minutes                    |
| Results Displayed | All Answers, Submitted Answers, Correct Answers |

## Question 1


10 out of 10 points

It's important for some classes to have exactly one instance. Although there can be many printers in a system, there should be only one printer spooler. There should be only one file system and one window manager. A digital filter will have one A/D converter. An accounting system will be dedicated to serving one company.


How do we ensure that a class has only one instance and that the instance is easily accessible? A global variable makes an object accessible, but it doesn't keep you from instantiating multiple objects.

A better solution is to make the class itself responsible for keeping track of its sole instance. The class can ensure that no other instance can be created (by intercepting requests to create new objects), and it can provide a way to access the instance.

What OODP is the text referring to?

Selected Answer:  Singleton

Answers:

- Branch
- Bridge
- Cache
- Flyweight
-  Singleton


All answers are wrong

## Question 2


10 out of 10 points

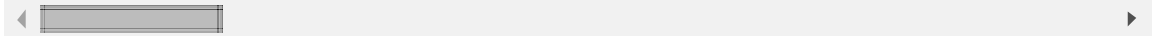
Trying to use objects at very low levels of granularity is nice, but the o

What OODP is the text referring to?

Selected Answer:  Flyweight

Answers:

- Branch
-  Flyweight
- Singleton
- All answers are wrong
- Cache
- Bridge



### Question 3

10 out of 10 points

The following Java class is a good example of a \_\_\_\_\_ OODP.


```
class ComputerComposite {
    private CPU processor;
    private Memory ram;
    private HardDrive hd;

    public ComputerComposite() {
        this.processor = new CPU();
        this.ram = new Memory();
        this.hd = new HardDrive();
    }

    public void start() {
        processor.freeze();
        ram.load(BOOT_ADDRESS, hd.read(BOOT_SECTOR, SECTOR_SIZE));
        processor.jump(BOOT_ADDRESS);
        processor.execute();
    }
}

/* Client */

class You {
    public static void main(String[] args) {
        ComputerComposite computer = new ComputerComposite ();
        computer.start();
    }
}
```

Selected Answer:  Facade

Answers:

- Bridge
- Flyweight

Singleton

Composite

 Facade

Prototype

---

## Question 4

10 out of 10 points

When an abstraction can have one of several possible implementations, the usual way to accommodate them is to use inheritance. An abstract class defines the interface to the abstraction, and concrete subclasses implement it in different ways. But this approach isn't always flexible enough. Inheritance binds an implementation to the abstraction permanently, which makes it difficult to modify, extend, and reuse abstractions and implementations independently.

You can overcome this issue by using an OODP. This OODP is used to decompose the abstraction's interface and implementation into orthogonal class hierarchies. The interface class contains an object reference to the abstract implementation class. This reference is initialized with an instance of a concrete implementation class, but all subsequent interaction from the interface class to the implementation class is limited to the abstraction maintained in the implementation base class. The client interacts with the interface class, and it in turn "delegates" all requests to the implementation class.

The interface object is the "handle" known and used by the client; while the implementation object, or "body", is safely encapsulated to ensure that it may continue to evolve, or be entirely replaced (or shared at run-time.)


Use the OODP when:

- you want run-time binding of the implementation,
- you have a proliferation of classes resulting from a coupled interface and numerous implementations,
- you want to share an implementation among multiple objects,
- you need to map orthogonal class hierarchies.

Consequences include:

- decoupling the object's interface,
- improved extensibility (you can extend, i.e. subclass, the abstraction and implementation hierarchies independently),
- hiding details from clients.

What OODP is the text referring to?

Selected Answer:  Bridge

Answers:  Bridge

Branch

Singleton

Cache

All answers are wrong


Flyweight

---

## Question 5

0 out of 10 points

The information about the external state of *Flyweight* objects is

Selected Answer:  Shared by some *Flyweight* objects.

Answers: Shared by some *Flyweight* objects.

Stored by clients of *Flyweight* objects.



Shared by all the *Flyweight* objects.

Shared by no *Flyweight* objects.

---

## Question 6


10 out of 10 points

The following Java class is an example of \_\_\_\_ OODP.

```
public abstract class C implements Cloneable{  
    public abstract Prototype clone();  
}
```

```
public class C1 extends C {  
    @Override  
    public C clone() {  
        return super.clone();  
    }  
}
```

```
public class C2 extends C{  
    @Override  
    public C clone() {  
        return super.clone();  
    }  
}
```

Selected Answer:  Prototype

Answers:

- Composite
- Flyweight
- ☒ Prototype
- Facade
- Bridge
- Singleton

---

## Question 7

10 out of 10 points

What is the OODP ?

```
class Client {  
    Component comp;  
}  
  
class Component {  
    public void op1 () {...}  
    public void op2() {...}  
}  
  
class Leaf extends Component {    public void op1 () {...}  
    public void op2() {...}  
}  
  
class Branch extends Component {  
    Set<Component> compSet;  
    public void op1 () {...}  
    public void op2() {...}  
}
```

Component represents a base class (or possibly an interface) for primitive objects, and Branch represents a “Branch” class. For example, the Component class might represent a base class for graphic primitives, whereas the Branch class might represent a Drawing class. Leaf class represents a concrete primitive object; for example, a Line class or a Text class. The Operation1() and Operation2() methods represent domain-specific methods implemented by both the Component and Branch classes. The Branch class maintains a collection of components. Typically, Branch methods are implemented by iterating over that collection

and invoking the appropriate method for each Component in the collection.

Selected Answer: ☒ Composite

Answers:

- ☐ Bridge
- ☒ Composite
- ☐ Flyweight
- ☐ Facade
- ☐ Prototype
- ☐ Singleton

---

### Question 8

10 out of 10 points

What design pattern would you say the following Java class is based on?

```
public class PrintSpooler {  
    private static PrintSpooler spooler;  
  
    private PrintSpooler() {  
    }  
  
    public static synchronized PrintSpooler getSpooler() {  
        if (spooler == null)  
            spooler = new PrintSpooler();  
        return spooler;  
    }  
    public void print(String s) {  
        System.out.println(s);  
    }  
}
```

Selected Answer: ☒ Singleton

Answers:

- ☐ Bridge
- ☐ Composite
- ☐ Flyweight
- ☒ Singleton
- ☐ Prototype
- ☐ Facade

---

### Question 9

0 out of 20 points

The follow Java class design is an example of \_\_\_\_ OODP.

```

abstract class Vehicle {
    protected Workshop workShop1;
    protected Workshop workShop2;

    protected Vehicle(Workshop workShop1, Workshop workShop2) {
        this.workShop1 = workShop1;
        this.workShop2 = workShop2;
    }

    abstract public void manufacture();
}

class Car extends Vehicle {

    public Car(Workshop workShop1, Workshop workShop2) {
        super(workShop1, workShop2);
    }

    @Override
    public void manufacture() {
        System.out.print("Car ");
        workShop1.work();
        workShop2.work();
    }

}

class Bike extends Vehicle {

    public Bike(Workshop workShop1, Workshop workShop2) {
        super(workShop1, workShop2);
    }

    @Override
    public void manufacture() {
        System.out.print("Bike ");
        workShop1.work();
        workShop2.work();
    }

}

interface Workshop {
    abstract public void work();
}

class Produce implements Workshop {

    @Override
    public void work() {
        System.out.print("Produced");
    }

}

```

```

class Assemble implements Workshop {

    @Override
    public void work() {
        System.out.println(" Assembled.");
    }

}

public class Pattern {


    public static void main(String[] args) {

        Vehicle car = new Car(new Produce(), new Assemble());
        car.manufacture();
        Vehicle bike = new Bike(new Produce(), new Assemble());
        bike.manufacture();

    }

}

```

Selected Answer:  Composite

Answers:  Bridge

Facade

Prototype

Composite

Flyweight

Singleton

Wednesday, November 18, 2015 6:50:04 PM EST