

# Quiz 1

1.

What is the output of the code below?

```
public class Employee {
    private String name;
    private float salary;

    public Employee(String name, float salary) {
        this.name = name;
        this.salary = salary;
    }

    public Employee changeName (Employee e, String name) {
        e.name = name;
        return e;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public static void main(String[] args) {
        Employee e = new Employee("bob", 30000);
        e.name = "Fred";
        e = e.changeName(e, "Bob");
        System.out.println(e.getName());

        Calendar hireDate = null;
        e = new Director("sam",10, hireDate);
        Manager m = new Director("sussan",10, hireDate);
        Admin a = new Admin("ad",10);
    }
}
```

<input type="radio"/>	Fred
<input checked="" type="radio"/>	Bob
<input type="radio"/>	null
<input type="radio"/>	empty string

2.

Suppose the following two classes are defined:

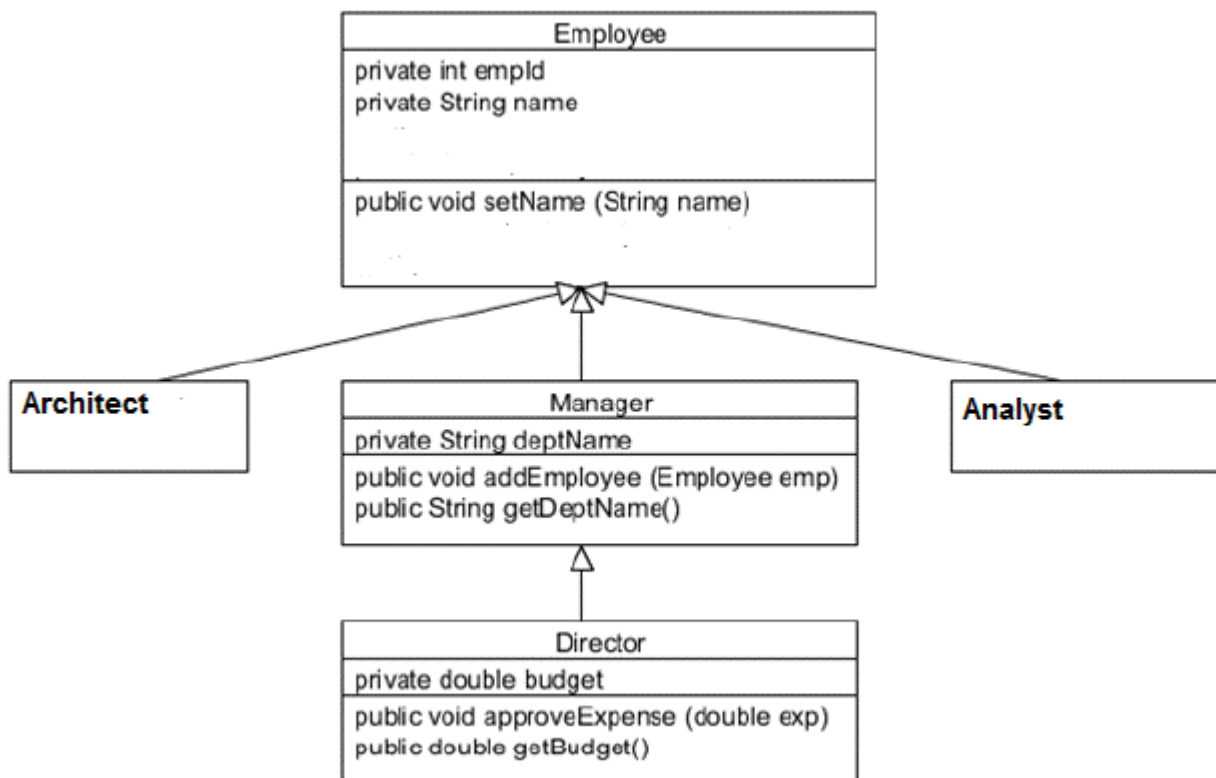
```
public abstract class Figure {...}
public class Rectangle extends Figure {...}
public abstract class 3DFigure extends Figure {...}
```

Which of the following instantiations is *not* valid?

<input type="radio"/>	Rectangle r = new Rectangle(.....);
<input type="radio"/>	Figure f = new Rectangle(....);
<input checked="" type="radio"/>	Figure f = new 3DFigure(...);

3.

Using the class diagram below, examine the code that follows and choose which of the following statements below does *not* compile?



```
Employee e = new Director();
Analyst a = new Analyst();
Manager m = new Director();
```

<input checked="" type="radio"/>	e.addEmployee()
<input type="radio"/>	m.addEmployee(a);
<input type="radio"/>	((Director)m).approveExpense(10000)

4.

The code below produces a compilation error. Examine the code and choose the fix that will enable the classes to compile

```
public class Account {
    private double balance;
    public Account (double balance) { this.balance = balance; }
    // other getter and setter for balance
}
public class Savings extends Account {
    private double interestRate;
    public Savings(double rate) {
        this.interestRate = rate;
    }
}
```

<input type="radio"/>	Call the setBalance method of the Account from Savings
<input type="radio"/>	Change the access of interestRate to public
<input type="radio"/>	Add a no-arg constructor to class Savings
<input checked="" type="radio"/>	Replace the constructor in Savings with one that calls the constructor of Account using super.

5.

Which of the following statements is *untrue* about an “immutable” class ?

<input type="radio"/>	All attributes must be private to prevent access from outside the class
<input type="radio"/>	Have a constructor that enables an object to be instantiated the first time with values
<input checked="" type="radio"/>	An object instance cannot be changed after it is created
<input type="radio"/>	Provide only setter and getter methods to access the attributes from outside the class

**6.****An abstract method must *not* have:**

<input checked="" type="radio"/>	a method implementation
<input type="radio"/>	a return value
<input type="radio"/>	method parameters
<input type="radio"/>	a protected access modifier

**7.****Which of the following is *untrue* about interfaces and inheritance?**

<input type="radio"/>	A class can extend multiple interfaces
<input checked="" type="radio"/>	An interface can extend multiple interfaces
<input type="radio"/>	A class can extend another class and implement multiple interfaces
<input type="radio"/>	All methods in an interface are implicitly abstract, unless provided with a default implementation

**8.****Which of the following statements is *untrue* about method overriding?**

<input type="radio"/>	Constructors cannot be overridden
<input type="radio"/>	If a static method in the base class, is redefined in the sub-class, the later hides the method in the base class
<input type="radio"/>	In method overriding, run-time polymorphism ensures that instantiated, the call to any method in the base class will be resolved to the correct method, based on the run-time type of the object instantiated.
<input checked="" type="radio"/>	During method overriding, the overridden method in the sub-class can specify a weaker access modifier

**9.****Choose the *incorrect* statement**

<input type="radio"/>	The principle of least knowledge reduces dependencies between objects and promotes loose coupling
-----------------------	---

<input type="radio"/>	The code below is a good example of the principle of least knowledge
	<pre>Driver driver = car.getDriver() Address driverAddress = driver.getAddress()</pre>
<input checked="" type="radio"/>	According to the principle of least knowledge, accessing the methods on objects returned by a method call is invalid
<input type="radio"/>	The principle of least knowledge states that accessing methods of objects passed in as parameters or instantiated inside the method is valid