

## Choosing Classes

1. Consider the following problem:

A company allows its employees to check out certain items, such as handheld computers and music players to gain personal experience with them. Popular items can be reserved on a first come/first served basis. A reservation list is kept for each item. There is a fine for overdue items.

Your development team's task is to write a software program that allows the stockroom clerk to check out items and check them back in, to reserve items, to notify employees when a reserved item has been returned, to produce reports of overdue items, and to track payment of fines.

What classes would you choose to implement this program?

## Cohesion and Coupling

2. Consider the `java.awt.Toolkit` class in the standard Java library. Is its public interface cohesive? Explain why or why not.
3. Which of the following classes depend on each other?  
`String`  
`StringTokenizer`  
`PrintStream`  
`Random`

Hint: Look at the API documentation.

## Accessor and Mutator Methods

4. Look up the methods of the `StringTokenizer` class in the API documentation. Indicate which methods are accessors, and which are mutators.

Hint: Remember that accessor methods do not change the state of the object. Therefore, if you call an accessor method multiple times in a row with the same parameters, you always get the same answer.

5. A class is immutable if it has no mutator methods.

List four immutable classes.

## Side Effects

6. The following class has a method with a side effect:

```
/**
 * A purse computes the total value of a collection of coins.
 */
public class Purse
{
    /**
     * Constructs an empty purse.
     */

    public Purse()
    {
        total = 0;
    }

    /**
     * Add a coin to the purse.
     * @param aCoin the coin to add
     */
    public void add(Coin aCoin)
    {
        total = total + aCoin.getValue();
        System.out.println("The total is now " + total);
    }

    /**
     * Get the total value of the coins in the purse.
     * @return the sum of all coin values
     */

    public double getTotal()
    {
        return total;
    }

    private double total;
}
```

Describe the side effect, and explain why it is not desirable.

7. How would you eliminate the side effect?

## Preconditions and Postconditions

8. What are the preconditions of the `nextLine` method of the `Scanner` class?

9. What happens if you violate one of those preconditions?
10. Coins should not have a negative value or a value of zero. Document an appropriate precondition of the `Coin` constructor to ensure these requirements.
11. Suppose a `CashRegister` class has the following methods:  
`recordPurchase`  
`enterPayment`  
`giveChange`  
`getPurchaseAmount`

Provide an appropriate postcondition for the `recordPurchase` of the `CashRegister` class, which is defined as follows:

```
/**
 * Records the purchase price of an item.
 * @param amount the price of the purchased item
 * Precondition: amount >= 0
 */
public void recordPurchase(double amount)
{
    purchase = purchase + amount;
}
```

## Static Methods

12. A static method has no implicit parameters. Sometimes, we use static methods because all the method parameters are numbers. Numbers are not objects, so they cannot be implicit parameters of a method.

Write two static methods that compute the circumference of a circle with a given radius `r` and the area of a circle with a given radius `r`. Place the two static methods into a class `Geometry`

13. An overuse of static methods is often a sign of poor object-oriented design. Explain how you can compute the circumference and area of circles in a more object-oriented fashion. Provide the more object-oriented solution.

## Static Fields

14. Consider the `Needle` class from Chapter 7 of the textbook.

Each needle object has its own random number generator object, which is wasteful. A single random number generator can be shared among all needle objects.

Reimplement the `Needle` class so that all objects share a `static` generator.

## Scope

15. Suppose you add the following method

```
/**
 * Compares the radius of another circle with the radius of
 * this circle.
 * @param radius the radius of the other circle
 * @return A value less than zero if the radius of this
 * circle is smaller than
 * the radius of the other circle, 0 if they are equal, and a
 * value
 * greater than zero if the radius of this circle is greater
 * than the
 * radius of this circle
 */
public int compareRadius(double radius)
{
    final double EPSILON = 1E-12;
    double diff = radius - radius;
    if (Math.abs(diff) < EPSILON) return 0;
    if (diff < 0) return -1;
    if (diff > 0) return 1;
}
```

to the `Circle` class you created in this lab (did you?). Note that the return statement has an error, because it is trying to use the radius of this circle and the radius of the other circle, but both have the same name (overlapping scope). How can we access the shadowed field name?

## Packages

16. Place the `Bank` and `BankAccount` programs of Chapter 8 into a package named `com.horstmann.java`.

Into which directory should you place the code files?

17. What modifications do you need to make to the files `BankAccount.java` and `Bank.java`?
18. What modifications do you need to make to the file `BankTester.java`?