Practice Final Examination

This is a practice exam that is similar in scope and style to the final exam that will be given on Thursday, December 17, 2009 at 1:00. Please note that this practice exam has extra questions on it, and parts of this practice exam may be tougher than the real exam. (This practice exam is much longer and tougher than the real exam.)

The final exam will be an open book/open notes exam and you will have 120 minutes to complete it. (No electronic devices are allowed.)

If you do not understand a question, please ask for a clarification. (Several of the problems come from previous exams. This practice exam is not comprehensive, I reserve the right to test other concepts.)

If you have a legitimate reason for missing the final exam, please contact me before the exam so we can make other arrangements.
1. Which one of these statements is **false**?
   A. A representation invariant places restrictions on the contents of one or more variables.
   B. Abstraction functions map a set of data values to an abstract value.
   C. The compiler enforces the restrictions specified by representation invariants.
   D. It is a good idea to mark the fields in a class **private** or **protected** to ensure that the representation invariant holds.

2. Consider the statement `double d = compute(x);` Which one of the following **cannot** be deduced from the statement:
   A. The `compute` method exists in 'this' class.
   B. The `compute` method takes one parameter.
   C. The `compute` method is not void.
   D. The `compute` method returns a **double** value.

3. Consider the looping statement `for (int i = 0; i < 10; i = i * 2)` Assuming the body of the loop does not break the loop or change `i`, how many times would the body of the loop be executed?
   A. 0
   B. 3
   C. 7
   D. 10
   E. ∞

4. Consider the looping statement `for (int i = x; i <= 10; i = i + 1)` Assuming the body of the loop does not break the loop or change `i`, how many times would the body of the loop be executed?
   A. 0
   B. `x`
   C. `10 - x`
   D. `11 - x`
   E. ∞

5. What is a try / catch block used for in Java?
   A. To throw exceptions when an error occurs
   B. To ignore exceptions so that they won’t crash a program
   C. To intercept exceptions so that errors can be handled appropriately
   D. To prevent exceptions from being thrown

6. Which one of these might throw an exception?
   A. Declaring a variable whose type is an interface
   B. Getting the `ith` value from an array of `String` objects
   C. Storing an `int` in a `long` variable
   D. Returning early from a `void` method

7. Consider the boolean expression `isMoon() && (isBlue() || isFull())` Under what condition(s) will `isFull()` **not** be called when this expression is evaluated? (Select all that apply.)
   A. `isMoon()` returns `true`
   B. `isMoon()` returns `false`
   C. `isBlue()` returns `true`
   D. `isBlue()` returns `false`
static public String doesSomething (String s) {
    String result = "";
    for (int p = 0; p < s.length(); p++)
        result = s.charAt(p) + result;
    return result;
}

8. What happens when the above method is called (pick only one)?
   A. A new string containing the first character in s is returned.
   B. A new string containing the characters in s in reverse order is returned.
   C. The order of the characters in s is reversed.
   D. An exact copy of the characters in s is returned.
   E. The reference to the String object s is returned.
   F. Nothing - an exception is thrown due to an error in the code.

9. Which one of these is false:
   A. Both s and result are local variables.
   B. Both s and result will be initialized before they are used.
   C. When the method above is run, s may be null.
   D. When the method above is run, this refers to the ‘current’ object.
   E. If no exception is thrown, this method will always return a non-null String object.

10. Because the method above is static, it would be illegal to make it private, true or false?

11. Consider the expression \((\text{int}) 42.6\), what value and type will be the result?
    A. A double equal to 42
    B. A double equal to 43
    C. An int equal to 42
    D. An int equal to 43

12. Consider another expression of the form \((\text{A}) \ b\), which one of the following is false?
    A. The expression is an example of a typecast.
    B. The compiler will view b as a value of type A in this expression.
    C. An exception may be thrown if b cannot be legally cast to type A.
    D. If b is an object, the type of the object stored in b is permanently changed to type A.

13. Which one of the following is false:
    A. A subclass inherits methods and data from a superclass.
    B. A subclass object (of some superclass) may be used any place the superclass object is expected.
    C. A superclass may override methods written in the subclass.
    D. A subclass may have methods not declared in the superclass.
14. Consider the following four statements that make use of inheritance and polymorphism. Assume classes Alpha and Beta are subclasses of class Gamma.

```
Alpha a = new Alpha ();
Beta b = new Beta ();
Gamma g = new Gamma ();
g = b;
// Next (fifth) statement goes here.
```

Consider the fourth statement \( g = b; \). Which of the following best describes the effect of this statement?

A. \( g \) gets a copy of the Beta object in \( b \).
B. \( g \) will refer to the same object as \( b \).
C. A new Gamma object is created from \( b \) and is stored in \( g \).
D. A new Beta object is created from \( g \) and is stored in \( b \).
E. A run-time exception is thrown.

15. Which one of the following statements, when viewed as the fifth statement, would be illegal at compile time?

A. \( a = new Gamma () \);
B. \( g = new Alpha () \);
C. \( b = (Beta) g; \)
D. \( g = a; \)
E. \( b = new Beta () \);

16. Which one of the these statements, when viewed as the fifth statement, would throw an exception at run time?

A. \( g = (Alpha) a; \)
B. \( g = (Gamma) b; \)
C. \( b = (Beta) g; \)
D. \( a = (Alpha) g; \)
E. \( g = a; \)

17. Select the comment below that is a javadoc formatted comment.

A. /**
   * This method causes the computer to emit a strange smell.
   */
B. // This method causes the computer to emit a strange smell.
C. \* This method causes the computer to emit a strange smell. */
D. /*
   * This method causes the computer to emit a strange smell.
   */
E. // This method causes the computer to emit a strange smell.

18. Select the best definition for polymorphism.

A. Data and methods are grouped together in a class.
B. One class may extend, or specialize, another class.
C. The user must not rely on the implementation details of a class.
D. The behavior of an object is determined by its content, not its context.
E. If class Alpha extends class Gamma, Alpha is a Gamma.
19. Which one of these will create an array containing 4 String elements?
   A. `String[4] s;`
   B. `String s = String[5];`
   C. `new String[] {"Mon", "Tue", "Wed", "Thu"}`
   D. `new String[5]`

20. What abstract concept does a `JFrame` object represent?
   A. A border around a graphical image
   B. A top-level window
   C. An object that can be used to draw to the screen
   D. The desktop

21. Suppose you wanted to add components to the north and south sides of a panel. Which layout manager would you choose?
   A. `FlowLayout`
   B. `GridLayout`
   C. `BorderLayout`
   D. `RowLayout`

22. How many primitive types are there in Java?
   A. 5
   B. 6
   C. 7
   D. 8

23. Which one of these best describes immutable objects?
   A. An object whose state cannot be changed
   B. An object without any ‘set’ methods
   C. An object without a constructor
   D. An object that represents an abstract value

24. A method contract specifies preconditions and postconditions, true or false?

25. Testing is the least important of the stages of programming, true or false?

26. In Java, it is legal to create an array of arrays, true or false?

27. An abstract data type specifies a specific implementation of how data must be stored and manipulated in a class, true or false?
For each of these questions, you are to write a public, static method that satisfies the given contract. You do not need to write an entire class, just write one method. You may select any reasonable names for your methods.

1. Write a method that fulfills this contract:

```java
/**
 * This method returns the sum of the largest and smallest values found
 * in an ArrayList. The method takes one parameter, an ArrayList of
 * Integers. If the ArrayList is empty, a runtime exception is thrown.
 * The return value is an integer equal to the sum of the largest and
 * smallest values found in the list.
 */
```

2. Write a method that fulfills this contract:

```java
/**
 * This method reverses the order of elements in an array. The method takes
 * one parameter, an array of Objects. Nothing is returned from this
 * method, but the array referenced by the parameter will be reversed.
 * A 'Reversed' array in this context means the value that was in position
 * 0 will now be found at the last position of the array,
 * the value that was at position 1 will now be found in the next to last
 * position, etc, and the value that was in the last position will now
 * be in position 0.
 */
```
Use if statements, expressions, local variables, and return statements to complete the method below so that it satisfies the following description.

The method below should compute and return a double that corresponds to the yearly salary for an employee. The method takes two parameters, an integer that corresponds to the employee’s years of experience, and a double that contains a base salary. (See the method header below.) You should calculate and return the salary for an employee using these rules:

- If the employee has 5 years experience or less, their salary is just the base salary.
- If the employee has more than 5 years experience, they receive a bonus of 10% of their base salary for each year of experience past 5 years.
  
  Example: If an employee has eight years of experience, she receives her base salary plus 30% of the base salary for her three extra years of experience.

- No employee should receive more than twice the base salary. This is the maximum salary that should ever be returned.
  
  Example: If an employee has 20 years of experience, he should get his base salary plus a bonus of 150% of the base salary as a bonus for his extra 15 years of experience. This would add up to be 250% of the base salary. Unfortunately, he can receive at most twice the base salary, so his salary is limited to 200% of the base salary.

Write statements inside the method below to complete the method. Your code will be graded for correctness and proper syntax. A typical solution will be 6 to 10 lines of code.

```java
/* Computes and returns the salary for an employee.
 * The parameter 'years' contains the employee's number of years of experience.
 * The parameter 'base' contains the base salary for the employee.
 */
public double calculateSalary(int years, double base) {
    // Your code here
}
```
The code below is a partial implementation of a Item class and a partial implementation of an Inventory class. You are to provide the code that would go in the blanks. (You must decide what instance variables are needed, and you must finish each method.) Most answers are either one or two lines of code but some take more, as indicated.

**Item class**

```java
public class Item {
    ________________________________; // Blank 5-1
    ________________________________; // Blank 5-2
    ________________________________; // Blank 5-3

    /** Creates an item with the specified description, price, *
     *  and quantity. */
    public Item (String description, double price, int quantity) {
        ________________________________; // Blank 5-4
        ________________________________; // Blank 5-5
        ________________________________; // Blank 5-6
    }

    /** This method returns the price of this item. */
    public double getPrice () {
        // You do not need to write this method, assume it has already been done.
    }

    /** This method changes the price of this item to the *
     * specified price. */
    public void setPrice (double newPrice) {
        // You do not need to write this method, assume it has already been done.
    }

    /** This method returns the number of these *
     * items that the store has in its inventory. */
    public int getQuantity () {
        // You do not need to write this method, assume it has already been done.
    }

    /** This method returns true if the item’s quantity *
     * is greater than 0. */
}
```
public boolean isInStock ()
{
    // You do not need to write this method, assume it has already been done.
}

/** Returns the description for this item */
public String getDescription ()
{

} // Blank 5-7

/** Returns the total value of the store’s inventory of this item (which is price times quantity). */
public double getTotalValue ()
{

} // Blank 5-8

/** Increases the quantity of this item by the specified number of items. */
public void increaseQuantity (int additionalItemCount)
{

} // Blank 5-9

/** If the item is 'in stock', this method decreases the quantity of this item by 1. */
public void decrementQuantity ()
{

} // Blank 5-10
}
/** An inventory will contain three items:
 *  item1 - the first item, or null if none
 *  item2 - the second item, or null if none
 *  item3 - the third item, or null if none
 */
public class Inventory
{
  private Item item1;
  private Item item2;
  private Item item3;

  /** Creates an inventory object without any items in it. */
  public Inventory()
  {
  // Blank 5-11
  // Blank 5-12
  // Blank 5-13
  }

  /** This method returns the specified item, or null if none. */
  public Item getItem(int itemNumber)
  {
    // You do not need to write this method, assume it has already been done.
  }

  /** This method adds the specified item to the
   *  the first empty position (if any) in the inventory. */
  public void setItem(Item newItem)
  {
    // You do not need to write this method, assume it has already been done.
  }

  /** Returns the number of items in this
   *  inventory, either 0, 1, 2, or 3. */
  public int getInventorySize()
  {
    // Blank 5-14
  }
}
/** Returns the total value of items in
 * this inventory by summing up the total
 * values of each item (if any).
 */
public double getInventoryValue ()
{

                        // Blank 5-15

}
A mobile is a hanging sculpture made from strings, rods, and bobs. The following two pictures are mobiles constructed by artists to balance interesting objects in interesting ways. In the first one, the bobs are butterflies. In the second, the bobs are various fictional characters.

For this problem you are to implement a mobile abstract data type. Abstractly, a mobile is a hanging sculpture made of strings, rods, and bobs and is either:

- A string that supports a bob. (The bob has a weight.)
- A string that supports a rod that has two simpler mobiles attached to its left and right ends. (The rod is supported at its center.)

The picture below illustrates a mobile that consists of five vertical strings, two horizontal rods, and three circular bobs.

Notice how the definition of a mobile is structurally recursive. (A large mobile can be thought of as a pair of simpler mobiles hanging from a rod.)

We can use this structural recursion to create classes that represent a mobile. There will be three classes: a Mobile interface that describes the abstract behavior of any mobile, a MobileRod class that represents one rod in a mobile (with other mobiles hanging from it), and a MobileBob that represents the simplest possible mobile, a bob hanging on a String.
The following interface is part of a mobile implementation:

// A Mobile is either a BobMobile or a RodMobile.

public interface Mobile {

    /**
     * Returns the total weight of this Mobile. Only bobs contribute to the weight
     * of a mobile; the strings and rods are massless. (The total weight of the
     * mobile shown above is 90; the total weight of its left submobile is 40 and of
     * its right submobile is 50.)
     */
    public int totalWeight();

    /**
     * Returns true if every rod in this Mobile will be level when the Mobile is
     * hung from the ceiling. Returns false otherwise. (The mobile shown above
     * is not balanced; however, both its left and right submobiles are balanced.)
     */
    public boolean isBalanced();

    /**
     * Modifies this Mobile by doubling the mass of each bob that it contains.
     * (After running this method on the Mobile above the masses of its three bobs,
     * reading left to right, would be 80, 50, and 50.
     */
    public void doubleAllMasses();

}

Using the classes and constructors that follow (see next pages), the mobile pictured on the previous page can be represented as:

    new RodMobile(new BobMobile(40), new RodMobile(new BobMobile(25), new BobMobile(25)))
1. **MobileBob.java**: Complete this implementation of `MobileBob`. Do not add any features or functionalities that are not required.

   // A MobileBob represents a Mobile that consists of a single bob hanging on a string. // The bob has a mass.

   ```java
   public class MobileBob implements Mobile {
     private int mass; // Mass of the bob

     /**
      * Initializes this MobileBob to be a bob with the specified mass.
      */
     public MobileBob (int mass) {
       this.mass = mass;
     }
   }
   ```

   Write your implementation of this class on the answer sheet. You do not need to rewrite the class header, field, or constructor from above. We will assume that what you write belongs here.

2. **MobileRod.java**: Complete this implementation of `MobileRod`. Do not add any features or functionalities that are not required.

   // A MobileRod represents a Mobile that consists of a straight rod that is hanging // by a string (attached to the center of the rod). The rod supports // two simpler Mobiles at its ends.

   ```java
   public class MobileRod implements Mobile {
     private Mobile left; // Mobile supported on the rod’s left end
     private Mobile right; // Mobile supported on the rod’s right end

     // Creates a MobileRod with the specified sub-mobiles.
     public MobileRod (Mobile left, Mobile right) {
       this.left = left;
       this.right = right;
     }
   }
   ```

   Write your implementation of this class on the answer sheet. You do not need to rewrite the class header, field, or constructor from above. We will assume that what you write belongs here.
For this problem, read through the Queue and Node classes below. These classes are the classes that were used as the starting point for programming assignment 14. Please note that comments and unneeded methods have been removed.

```java
public class Queue<E>
{
    private Node head;
    private Node tail;
    private int size;

    // There was one constructor and five methods in this class. Assume
    // they are still here, but that they don't matter.

    /****** You will write a method that goes right here. *******/

    private class Node
    {
        private E data;
        private Node next;

        public Node (E data)
        {
            this.data = data;
            this.next = null;
        }

        public E getData ()
        {
            return data;
        }

        public Node getNext()
        {
            return next;
        }

        public void setNext(Node n)
        {
            next = n;
        }

        public boolean containsData (E data)
        {
            if (this.data == null || data == null)
                return this.data == data;
            else
                return this.data.equals(data);
        }
    }
}
```
Write a method called **swap** that has the following API description:

```java
public void swap (E a, E b)
```

Given two data elements `a` and `b`, this method finds the elements in the list and swaps their positions in the list.

If element `a` is not in the list, nothing happens. If element `b` is not in the list, nothing happens. If element `a` is the same as `b`, nothing happens.

You may assume that `a` and `b` will be non-null.

If there are multiple occurrences of `a` and/or `b` in the list, only the first occurrence of `a` and the first occurrence of `b` will be swapped.

Here are a few diagrams for your convenience. Note that they would not normally appear on the final exam - but this is a practice exam.

An empty queue.

A queue with four elements in it.