What is your name?:  

There are two sections:

I. True/False  
   40 points; (2 points each, 20 questions)

II. Problems  
   60 points; (6 points each, 10 questions)

--------

100 points total

This test is worth 15% of your final grade. This test is open book and open notes. Use the back of the sheet if you need more room to write, although you shouldn’t need to. State your assumptions wherever it makes a difference. You have exactly 50 minutes.

I. True False: (2 pts. each)

1. Strings in Java are immutable. A concrete class extending an abstract class must give definitions for all abstract methods from the abstract class.

2. C/C++ arrays don’t know their size, but arrays in Java do. This is because arrays in Java are an ADT, but arrays in C/C++ aren’t.

3. A class is abstract only if all its methods are abstract.

4. If Dog and Cat are classes that extend class Animal and you want to store both Dogs and Cats in the same array, that array must be declared as type Object.

5. Assume Hamster and Turtle are classes that extend class Animal. Furthermore assume that Hamster and Turtle both have a sound() method that displays the sound each of those animals make, but class Animal does not have a sound() method. An array that contains instances of both Hamster and Turtle can exploit polymorphism to call the correct sound() method for each type of object in that array.

6. Regardless of the values in two class instances A and B,

   A.equals(B) will always behave the same as B.equals(A)

   If one is null, they are different.

7. Each instance of a class has its own dispatch vector.

8. When implementing an interface you only have to give definitions for those methods declared as abstract. They aren’t declared as abstract in an interface.

9. A try block can have multiple corresponding catch blocks. It is a good idea to catch type Exception first as a safety precaution.

10. Comparable is an interface, not a class. An array of type Comparable can be created.

11. Interfaces help give some of the characteristics of multiple inheritance to Java.

12. Java does not have stand-alone functions.
13. Parameters can be in, out, or in-out parameters. Java parameters are in-out parameters. They are in parameters.

14. Java parameters are by their very nature reference parameters. They are value parameters.

15. An interface in Java gives both the declarations and definitions for method, however it omits the body of the methods. Interfaces give only the declarations.

16. A method can be used to return a reference to a private data member. This can be useful to allow for chaining multiple operators together in a single statement. e.g., `Employee.getdate().setDay(5);`

17. A static method can reference its implicit argument by using `this`.

18. A private static instance variable in a class can be initialized even though there are no instances of the class.

19. Abstraction by Specification can be hard-coded into programs by careful use of assert statements. It is done use precondition & postcondition comments.

20. Java allows a constrained form of goto by allowing a break or continue statement to branch to a label given in the code. Those statements don't branch to a label, but rather indicate which loop is to be continued or broken out of.

II. Problems: (6 pts. each) Briefly answer each of the following.

1. Consider the following loop in ALGOL 60, as discussed in class.

   ```algol
   i := 1;
   for count := 1 step count until 3 * i do
      i := i + 1
   ```

   Give the equivalent code in a single for loop in Java:

   ```java
   int i = 1;
   for (int count = 1; count < (3 * i); count += count) {
      i = i + 1
   }
   ```

2. Memory can be allocated in three different ways, corresponding to three different "areas" of memory. Give the name of each.

   a) Static, e.g., the compiled program & static variables

   b) On the stack, e.g., parameters and local variables for a method call

   c) On the heap, e.g., every time we use new
3. Assume we want to pass a large object as the argument to a function call, where we do not want this object to be modified within the function.

   a) What is the main disadvantage of passing this large object by value?

   It takes a lot of room on the stack.

   b) What can be done to try and solve this problem?

   Pass by constant reference.

4. A method can return a reference to a private instance variable. Why could this be a problem?

   It can provide public access to what would otherwise be private data, violating the Information Hiding principle of OOP.

5. Consider the following code written using C syntax:

   ```c
   int value;  /* Global variable */
   int list[2];  /* Global array of two members, subscripted by 0 & 1 */

   void swap (int &a, int &b)
   {
     int temp = a;
     a = b;
     b = temp;
   }

   void main()
   {
     value = 1; list[0] = 3; list[1] = 0;
     swap( value, list[value] );
     swap( list[0], list[1] );
   }
   ```

   What are the values of the variable "value" and the array "list" at the end of the above program if parameters are passed by name?:

<table>
<thead>
<tr>
<th></th>
<th>list[0]</th>
<th>list[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2nd call</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
6. Consider the following classes:

```java
Class myDate {
    int day; int month; int year;

    public myDate( int m, int d, int y) {
        month=m, day=d; year=y;
    }

    public void setYear( int y) {
        year=y;
    }

    public String toString() {
        return month + "/" + day + "\"/" + year;
    }
}
```

```java
Class Employee {
    private String name;
    public myDate hireDate;

    public Employee() {
        name = "John Doe";
        hireDate = new myDate(9,29,2004);
    }

    public Employee( Employee e) {
        name = e.name;
        hireDate = e.hireDate;
    }

    public String toString() {
        return name + ", " + hireDate;
    }
}
```

Assume the following code is in a class in the same package as the above two classes.

```java
// ... other stuff
Employee e1 = new Employee();
Employee e2 = new Employee( e1);
e1.hireDate.setYear(2000);
System.out.println( e2);
```

What is the output of running this code?

```
John Doe 9/29/2000
```

7. Assume class C extends class B, and class B extends class A. Explain how we can get a constructor in class C to call the constructor in class A.

```
A
  |
  v
B
  |
  v
C
```

We cannot write `super.super()` in class C. Rather, in class C our constructor has `super()`, and then again in class B our constructor has `super()`.

8. In creating your own exception class, how do you decide which superclass to extend?

You must consider whether or not your Exception will be checked. Most likely it will be, so you then extend one of the checked exception classes.
9. Assume there is a Java interface you need to use, and that interface has many methods, though you only need to use one. You are concerned with cluttering up your code. What approach can you use to solve this problem? Create a concrete Adapter class that provides definitions (i.e., the body) for all of the interface methods. Then your class can extend this adapter class, overriding any methods you actually want to use.

10. Consider a set of classes Person, Student, BasketballPlayer, RugbyPlayer, Administrator and Address. All student athletes must be students, and to be a student you must also be a person. Each person also has an address. In addition, there are a special set of actions that can be performed on part-time students only. Draw the UML diagram for this set of classes.

PartTime is an interface, and as such the methods it has might be useful other places, such as part-time Administrator (but not PartTime Person!).

PartTime Interface

Person

Address

Student

BasketballPlayer

RugbyPlayer

Administrator

isa

isa

 isa