4th G51PRG self-test

1. Write a constructor for the Employee class which sets all the instance fields to the given parameters. Use the constructor of the superclass to set the first four fields.

   Answer:

   ```
   public Employee(String f, String l, String d, String a, String p){
      super(f, l, d, a);
      payroll = new String(p);
   }
   ```

2. Does `Employee` class have to provide an implementation for the methods `firstName()`, `lastName()`, `dateOfBirth()`, and `address()`?
   Answer: no, it does not have to. The method definitions are inherited from `SimpleFormFiller`.

   In what follows, assume that the class definitions are correct.

3. Suppose that the following main() function has been added to Employee.
   Would it compile?
   If yes, what will be printed after it is executed?

   ```
   public static void main(String[] args){
      Employee tom = new Employee("Tom", "Smith", "1980", "Unknown", "1111");
      Employee bob = new Employee("Bob", "Jones", "1970", "London", "1112");
      tom.companyName = "NewCompany";
      System.out.println(bob.companyName);
   }
   ```

   Answer: "NewCompany". `companyName` is static and therefore shared by `tom` and `bob`.

4. Can we invoke `dateOfBirth()` method on `tom` (declared as above)?
   Answer: yes, `tom` is an `Employee` and `Employee` inherits the method from `SimpleFormFiller`.

5. In the main() from question 3, suppose `tom` was declared as a `FormFiller`:

   ```
   FormFiller tom = new Employee("Tom", "Smith", "1980", "Unknown", "1111");
   ```

   Would this line (above) cause a compiler error?
   Answer: no, `tom` is an `Employee` hence (above in the hierarchy) a `SimpleFormFiller` and a `FormFiller`.

6. Tick all the lines which would cause a compiler error:

```
public static void main(String[] args){
   SimpleFormFiller tom = new Employee("Tom", "Smith", "1980", "X", "1111");
   Employee bob = new Employee("Bob", "Jones", "1970", "London", "1112");
   bob.companyName = "Bob's Company";
   tom.companyName = "NewCompany"; <- compiler error
}
```

Answer: the last line. Fields can be called only if the declared type has them. Declared type of `tom` is `SimpleFormFiller` which does not have a companyName field.

7. Would the compiler report an error for the main() below?
   If not, would Tom's payroll number be printed?

```
public static void main(String[] args){
   SimpleFormFiller tom = new Employee("Tom", "Smith", "1980", "X", "1111");
   tom.printForm();
}
```

Answer: no error; Tom's payroll number will be printed since the Employee's version of the method will be used.

8. Would the compiler report an error for the main() below?

```
public static void main(String[] args){
   SimpleFormFiller tom = new Employee("Tom", "Smith", "1980", "X", "1111");
   tom.payroll();
}
```

Answer: yes; `tom` needs to be cast to `Employee` first since `SimpleFormFiller` does not have `payroll()` method.

9. Consider the following definitions:

```
interface MyInterface {
   public Object getData();
}

class MyClass implements MyInterface {
   String name;
   public String getData() { return name;}
}
```

Would it compile?
Answer: no, `getData()` should have exactly the same return type in the class which implements the interface (exactly the same signature).

10. Write the implementation for `hasGreaterVolume()` which returns `true` if the current object has strictly greater volume, and `false` if it has the same volume or less.

```
public abstract class Vessel {
   public abstract int getVolume();
   public boolean hasGreaterVolume(Vessel v){
      return (this.getVolume() > v.getVolume());
   }
}
```