CS61B Summer 2006

Exam 1

11 July 2006

Your name:

Your login: cs61b-_____

This exam is worth 25 points (out of a possible 230 for your course grade).

This exam is open book, open notes - you may use any references on paper that you wish, but no electronic devices (e.g. laptop computers) are permitted.

This booklet contains 11 numbered pages (including this page). Please make sure that you have all of the pages before you begin. You will have 80 minutes to complete the exam.

Please turn off your cell phone now.

SOLUTIONS

1 What does Java output? (6 points)

There is a sheet attached to the back of the exam that you may tear off - the questions in this section will refer to code that is on that sheet.

What will the output be? Write on the line to the right of the code fragment. If it's an error, just write ERROR - you don't have to give the text of the error message. Suppose each code fragment is inside `main`. Each question is independent of the others (i.e. do not assume that the code from the previous questions has already run).

1.
```
int i = 0 ;
while i < 7 {
    if(i % 2 == 0) {
         continue;
    }
    System.out.print(i);
    i = i + 1;
}
```

   We decided to throw out this question due to the syntax error of there not being parens around the (i ¡ 7), and because asking for the output of something that doesn't actually terminate is too confusing. So everyone got a point.

2.
```
MyClass a = new MyClass(2);
MyClass b = new MyClass(4);
b = a;
a.setVar(17);
System.out.println(b.getVar());
```

   17 (a and b refer to the same object)

3.
```
MyClass[] ma = new MyClass[4];
ma[0].sayStuff();
```

   ERROR; ma[0] is null, so trying to call a method on it will give a null pointer exception. Creating the array does not create any MyClass objects.

4.
```
MyClass x = new MyClass(3);
MyClass y = new MyClass(3);
if(x == y) {
    System.out.println(``YES'');
} else {
    System.out.println(``NO'');
}
```

   "NO". x and y are not == since they do not refer to the same object.

5.
```
try {
   throw new MyEx2();
}
catch (MyEx1 a) {
     System.out.println(``catch1'');
}
catch (MyEx2 b) {
     System.out.println(``catch2'');
}
```

   "catch1", since every MyEx2 is also a MyEx1 (because MyEx2 extends MyEx1). "ERROR" was also an acceptable answer - it turns out that the compiler is actually smart enough to notice this kind of programmer error and warn you about it - we didn't know that until now!

6.
```
MyClass m = new ClassTwo();
m.sayStuff();
```

   "Hi from ClassTwo". Dynamic method lookup means that the version of the method called at runtime depends on the type of the actual object, not on the static type of the reference.

2 Multiple Choice (5 points)

Circle the letter for the correct answer. There is exactly one correct answer for each question.

1. How do you permanently change the static type of a variable?

   (a) casting
   (b) assignment
   (c) overriding
   (d) ** It's not possible.

   The static type of a variable cannot be changed. Casting only tells the compiler to treat a certain variable in a different way temporarily.

2. A class declared as `final`...

   (a) can't extend another class
   (b) ** can't be extended
   (c) can't be insantiated
   (d) none of the above

3. The value of a `static` field...

   (a) ** is the same for every instance of the class.
   (b) can never be changed.
   (c) must be set every time the class is instantiated.
   (d) none of the above

   Anything `static` does not require an instance of the class to be accessed - there is only one for the whole class.

4. Consider the following code:

```
public class Test {

  public void method1() {
     int x = 37;
     method2(x);
  }

  public void method2(int i) {
     method3(i);
     System.out.println(``here we are'');
  }

  public int method3(int j) {
     return j + 1;
  }

  public static void main (String[] args) {
      method1();
  }
}
```

We compile and run `Test`. When "here we are" prints, the variable x is...

(a) alive and in scope.

(b) ** alive, but not in scope.

(c) not alive, but in scope.

(d) neither alive nor in scope.

During the time that the printing is happening, method2 is on top of the stack, with method1 below it. So x is still alive (on the stack), but not in scope (not at the top of the stack).

5. Consider the following code:

```
public class K { }

public class F {
   public K k;
}

public class Test1 {
    public static void main (String[] args) {
          F f = new F();
          F[] fa = new F[2];
          fa[0] = f;
    }
}
```

When we run `Test1`, how many objects does `main` put on the heap?

(a) 0

(b) 1

(c) ** 2

(d) 3

(e) more than 3

main creates one `F` object and one `F[]` object. No `K` objects are created (note that we never said `new K()`).

3 Inheritance Design (3 points)

Given 2 things, say how they would be related if we were writing a Java program to represent them. Write the letter of the appropriate description next to each question.

- a. A has an instance variable of type B
  b. A extends B
  c. B has an instance variable of type A
  d. B extends A

1. A: Kitchen B: House
   House HAS-A Kitchen, so c.
2. A: Fruit B: Apple
   Apple IS-A Fruit, so d.
3. A: Restaurant B: Menu
   Restaurant HAS-A Menu, so a.

4 Identifying Errors (5 points)

Each of the following programs has an error in it and will not compile. For each program, describe in **one short phrase or sentence** what the error is.

1.
```
public class SuperException extends Exception {  }

public class SubException extends SuperException { }

public class Class1  {
   public void someMethod() throws SubException {
        System.out.println(``this is Class1'');
   }
}

public class Class2 extends Class1  {
   public void someMethod() throws SuperException {
      System.out.println(``this is Class2'');
   }
}
```

   When a child class overrides a method from its parent class, the new version of the method must declare an exception that is the same as or a superclass of the exception declared in the original version of the method. This is because anywhere we are expecting an object of type Class1, we can use an object of type Class2, so Class2 can't throw exceptions that Class1 doesn't.

2.
```
public class SomeClass {
   private int theInt;

   public static void main (String[] args) {
        System.out.println(``The int is `` + theInt);
   }
}
```

   The `theInt` variable is non-static, so it can't be referred to in the static `main` method.

3. ```
   public class MyClass {
      private String message;

      public MyClass(String s) {
         message = s;
      }
   }

   public class Test {
      public static void main (String[] args) {
          MyClass c = new MyClass();
      }
   }
   ```

   Since we wrote a constructor for MyClass that takes an argument, there is no longer a zero-argument default constructor, so `new MyClass()` won't work.

4. ```
   public class MyClass {
      private String message;

      public MyClass(String s) {
         message = s;
      }
   }

   public class Test {
      public static void main (String[] args) {
          MyClass c = new MyClass(``Hello'');
          System.out.println(c.message);
      }
   }
   ```

   The instance variable `c` is private and therefore can't be referred to outside of MyClass.

5.
```
public class SomeException extends Exception { }

public class MyClass {

   public void doStuff(int i) throws SomeException  {
       if (i < 10) {
           System.out.println(``i less than 10'');
       } else {
           System.out.println(``i not less than 10'');
           throw new SomeException();
       }
   }

   public void doMoreStuff() {
       System.out.println(``hello'');
       doStuff(3);
   }
}
```

   The `doMoreStuff()` method calls `doStuff(3)`, so it has to either put the call to doStuff in a try/catch block that catches SomeExceptions, or declare that it can throw a SomeException itself.

5 Programming (6 points)

We're writing a program to represent students. Two kinds of students are highschoolers and college students. So far, we've written classes to represent students in general and high school students. High school students are like regular students, except that when they study, they complain about SATs instead of math.

```
public class Student {
   public float gpa;

   public void study() {
       System.out.println(``Math is hard!'');
   }

   public void procrastinate() {
       System.out.println(``wasting time...'');
   }
}

public class Highschooler extends Student {

   public void study() {
       System.out.println(``I hate SATs!'');
   }
}
```

1. College students are like regular students, except for two differences. First, they have to know which dorm they live in (represented by a String). Second, they study like other students, but procrastinate first. Write the `CollegeStudent` class here. (3 points)

```
public class CollegeStudent extends Student {

   public String dorm;

   public void study() {
       procrasinate();
       super.study();
   }
}
```

That they have to know which dorm they live in is a new piece of state, so it's represented by a new instance variable. Studying like other students except for procrastinating first translates to calling `procrastinate()`, then the study method from the Student superclass. Some people called `super.procrastinate()` instead of `procrastinate()`, which was also fine. (It's probably slightly better to use just `procrastinate()`, in case college students procrastinate in some unique way, but this isn't terribly important either way.) Some people also wrote new constructors for CollegeStudent that took a dorm as an argument, or gave dorm a default value; these were both not required but fine to do.

2. Graders need to calculate average grades. Write the `calcAverage` method inside the `Grader` class that takes in an array of students and returns their average GPA. You must fill in the blank space in the method declaration and the body of the method. (3 points)

```
public class Grader {

   public float calcAverage(Student[] sa) {
        float total = 0.0;

        for (int i = 0; i<sa.length; i++) {
             total = total + sa[i].gpa;
        }

        return (total \ sa.length);
   }
}
```

The key points here were using a `Student[]` as the argument (realizing that you can ask all kinds of students for their GPA in the same way), looping over each element in the array while adding to a total, and realizing that you have to actually get the gpa from each student (saying `sa[i].gpa` instead of just `sa[i]`. Some people put the `float total = 0.0` line inside the for loop, which will reset the "total" to zero every time, so the sum won't actually be calculated.

Reference page for section 1:

```
public class MyClass c {
   private int theInt;

   public MyClass() { theInt = 0; }

   public MyClass(int i) { theInt = i;  }

   public int getVar() { return theInt;  }

   public void setVar(int i) { theInt = i; }

   public void sayStuff() {
        System.out.println(``Hello from MyClass!'');
   }

}


public class ClassTwo extends MyClass {

   public ClassTwo() { }

   public ClassTwo(int i) {
       super(i);
   }

   public void sayStuff() {
         System.out.println(``Hi from ClassTwo!'');
   }
}


public class MyEx1 extends Exception {  }

public class MyEx2 extends MyEx1 {  }
```