CS180 Spring 2010 Exam 1 Solutions, 15 February, 2010
Prof. Chris Clifton

Turn Off Your Cell Phone. Use of any electronic device during the test is prohibited.

Time will be tight. If you spend more than the recommended time on any question, **go on to the next one**. If you can't answer it in the recommended time, you are either going in to too much detail or the question is material you don't know well. You can skip one or two parts and still demonstrate what I believe to be an A-level understanding of the material.

Scoring descriptions given in the short answer section are just a rough guide.

Multiple Choice

Provide answers on the Scantron card. (You may mark them on this sheet as well, but the mark on the Scantron card is the official version.)

1 Method Signatures and Overloading (3 minutes, 2 points)

Given the following class:

```
public class BaseClass
{
  double add(int a, int b) { return a + b; }
  int add(int a, int b) { return a + b; } // I.
  double add(double a, double b) { return a + b; } // II.
  float add(int a, int b) { return a + b; } // III.
  int add(int a, int b, int c) { return a + b + c; } // IV.
  int add(double a, double b, double c) { return (int)(a + b + c); } // V.
}
```

Not all of the methods are acceptable - some are proper *overloading* of the add routine, but others are not compatible and will result in an error. Which of I., II., III., IV., and V. must be **removed** for the above class to compile and be usable.

A III, V

B IV, V

C **I, III**

D I, II, III

E I, II, III, IV

The signature of a method is based on the name and the number and type of the parameters (but not the type of the result). Two methods of the same name, but with different number or type of parameters, are different methods (overloaded).

2 Inheritance (2 minutes, 2 points)

What is the output of the following code:

```
public class BaseClass
{
   public void TheMethod() { System.out.println("In base class"); }
}

public class ChildClass extends BaseClass
{
   public void TheMethod() { System.out.println("In child class"); }
}

public class Driver
{
   public static void main(String[] args)
   {
      BaseClass c = new ChildClass();
      c.TheMethod();
   }
}
```

A In base class

B **In child class**

C In base class
In child class

D In child class
In base class

E "BaseClass c = new ChildClass();" will cause a compile error because the type is different.

The (non-static) method in the child class overrides *the one in the parent class; any use of that instance uses the method from the child class.*

3 Inheritance/interfaces (3 minutes, 1 points)

Which of the following is **true**:

A A child class can access all public, protected and private variables and methods of the base class.

B Let "Queue" be an interface, "Queue q = new Queue();" creates an instance of the Queue interface.

C **The "super" keyword is used to access variables and methods of the base class.**

D The "this" keyword can be used in a static method.

E **A static method cannot access an instance variable.**

A static method can be called without an instance being created. In such case, accessing the instance variable wouldn't make sense.

4 Primitive Types (1 minute, 1 point)

Which of the following is NOT a primitive type:

A double

B **String**

C int

D byte

E All are primitive types.

A primitive type is held directly in the variable; it is small and of fixed size. Strings are variable size and thus not primitive.

5 Concurrency (2 minutes, 1 point)

To create and use a new thread in a program we:

A Create a class that inherits from the Thread class.

B Override the run() method of the Thread class.

C Use the start() method to start the thread.

D Use the join() method to wait for the thread to exit.

E **All of the above.**

6 Interfaces (4 minutes, 2 points)

The following definition of an abstract "light switch" data type will be used for questions 6 and 7.

```
 interface LightSwitch {
  boolean ON = true;
  boolean OFF = false;

  void turnOn();   // Turn the light on.
  void turnOff();  // Turn the light off.
  boolean isOn();  // return ON if the light is on, OFF if it is off.
}
```

The following variables have also been declared:

```
  LightSwitch light;
  boolean status;
```

At some later point in the code (after variables have been instantiated, some method calls have been made, etc.), you see the following statements. Which one can you be certain will **not** work (will always cause either a compile or run-time error):

A light.turnOn();

B if (status == light.ON) { System.out.println("status is ON."); }

C **light = new LightSwitch();**

D LightSwitch l = light;

It is not possible to create an instance of an interface – this wouldn't make sense, since it isn't implemented.

7 Implementing Interfaces (4 minutes, 3 points)

The following class is a special light switch that supports dimming the light; full on is intensity 1, off is intensity 0.

```
class DimmerSwitch implements LightSwitch {
  double intensity = 0.0; // Invariant:  0.0 <= intensity <= 1.0

  public void dim(double intensity) {
    // If the given intensity is legal, set the intensity to the given value.
    if (0.0 <= intensity && intensity <= 1.0)
      this.intensity = intensity;
  }

  public void turnOn() { // Turn the light on.
    intensity = 1.0;
  }

  public boolean isOn() { // return ON if the light is on, OFF if it is off.
    if (intensity == 0) return OFF;
    else return ON;
  }
}
```

The above code will not compile. This is because:

A There is no constructor defined for the class.

B The method dim is not part of the interface.

C **The method turnOff has not been defined.**

D The method turnOn is not needed, we could just use dim(0.).

E This is a trick question, the code *will* compile.

To implement an interface, you must implement all of the methods included in the interface.

8 Class/Instance Variables (3 minutes, 2 points)

Given the following code:

```
public class MyClass
{
   private int time = 1;

   public static void main(String[] args)
   {

   }
}
```

Which of the following code is correct to put in the main() method.

A System.out.println(time);

B System.out.println(MyClass.time);

C MyClass inst = new MyClass();
System.out.println(time);

D **MyClass inst = new MyClass();**
System.out.println(inst.time);

E MyClass inst = new MyClass();
System.out.println(MyClass.time);

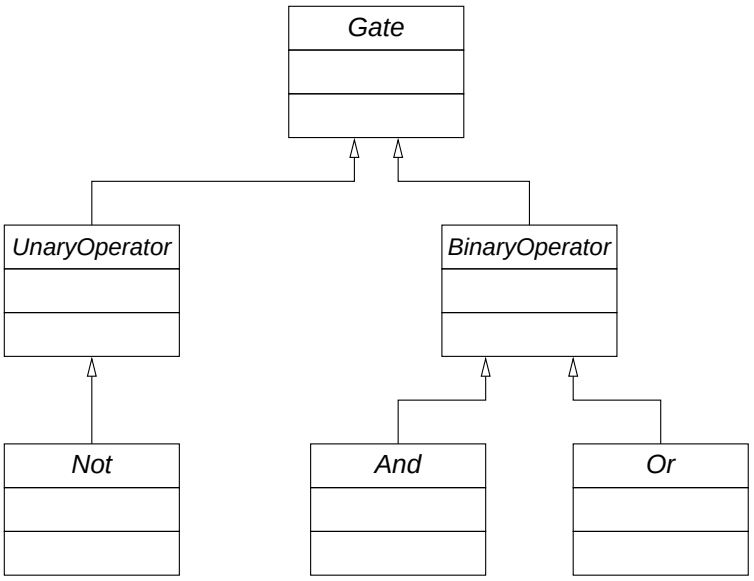
Static functions cannot access instance variables. Instance variables don't exist until the instance is created.

Short Answer

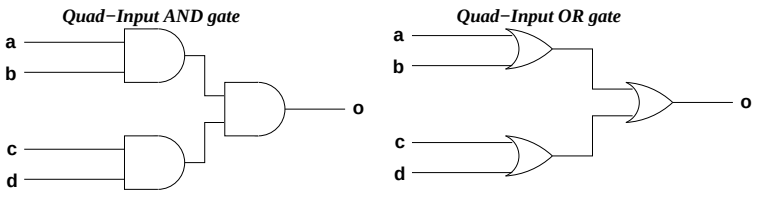
Write your answers in the space provided.

9 Programming with Generics, Inheritance (16 minutes)

This builds on the Boolean Circuits class hierarchy we discussed in lecture. (*Concurrent Programming* sections 7.1 and 7.4). *Hint: If you think you need to look at the book to solve this, you are missing the point of data abstraction. The only thing you need to know is the inheritance hierarchy:*



A QuadOperator is like a BinaryOperator, but it takes four inputs. Four-input AND and OR gates can be built (schematically) as follows:



A partial implementation of QuadOperator is:

```
class QuadOperator<T extends BinaryOperator> extends Gate {
  // Builds a four-input gate corresponding to the gate type T.

  T top, bottom, right;  // Upper, lower, and right gates as in preceding diagram.

  public QuadOperator () {
    super("Quad");
    // Additional code to instantiate top, bottom, right --
    // don't worry about how this is done.
    right.setOperand1(top);
    right.setOperand2(bottom);
  }

  public void setOperands(Gate a, Gate b, Gate c, Gate d) {
    top.setOperand1(a);
    // Code for the rest
  }

  public boolean getValue() {
    return right.getValue();
  }
}
```

You will need to perform some operations involving QuadOperators. If you aren't sure, give your best guess - there will be partial credit. Note that even if you get one part wrong, you can get the other parts right.

Hint: You really don't need to understand boolean circuits to answer these questions.

9.1 Variable Declaration (3 minutes, 3 points)

Write the code to declare (e.g., int i; a variable qand that is a four-input AND gate, and a variable qor that is a four-input OR gate.

QuadOperator<AND> qand;
QuadOperator<OR> qor;

Scoring: 1 for QuadOperator, 1 for <AND> / <OR>, 1 for syntactically correct.

9.2 Instance Creation (3 minutes, 3 points)

Write the code to create an instance of a four-input AND gate and assign it to the variable qand.

qand = new QuadOperator<AND>();

Scoring: 1 for calling constructor, 1 for <AND>, 1 for getting everything correct.

9.3 Use of an instance (4 minutes, 3 points)

You are given four gates a, b, c, and d:

```
Gate a,b,c,d;
a = new // ... Assume a, b, c, and d have been instantiated, don't worry about how.
```

Write code that uses the four-input AND gate qand to compute a && b && c && d, and assign the result to a boolean variable o. You should assume that the Gate, AND, and QuadOperator classes are completely implemented and work properly.

qand.setOperands(a,b,c,d);
boolean o = qand.getValue();
Scoring: 1 for calling setOperands, 1 for calling getValue, 1 for getting all details right (boolean o = , ...)

10 Fields and Inheritance (12 minutes)

Given the following code:

```
class Parent {
  private String s = "ParentString";

  public void printS () {
    System.out.println(s);
  }

  // Several other method implemented as well.
}
```

```
class Child extends Parent {
  private String s = "ChildString";

  // Several methods implemented, but there is
  // NO implementation of printS().
}
```

10.1 Inheritance and fields (3 minutes, 2 points)

Does the statement Parent p = new Child(); result in one string or two? Explain.

Two. Instantiating a child class creates the fields (and enables methods) of both child and parent classes. While methods can be overridden, fields are not.

Scoring: One point for getting it right (two), one for explaining that this creates an instance that has both parent and child fields. Possibly one point for the wrong answer but solid description of overriding.

10.2 Inheritance and fields (3 minutes, 2 points)

What would be the output of the following? Explain what is happening.

```
Parent p = new Child();
p.printS();
```

ParentString . Since the child class does not override the printS() method, this uses the parent's printS() method. Since fields are not overridden, it uses the string s from the parent.

Scoring: 1 for correct answer (ParentString), 1 for explanation either noting that fields are not overridden, or (incorrectly) describing the mechanism of overriding.

10.3 Field access (3 minutes, 3 points)

Suppose that in a method in class Child, you would like to change the value of the String s in class Parent. Either describe a way this could be done, or explain why it cannot be done.

Since the field is is private, we can't use super.s to access the field. Instead, we would need to call methods in Parent that would modify s (if such exist.)

Scoring: 1 for (improper) use of super, 2 for noting that because is private, it can't be accessed directly, 2-3 for suggesting use of methods in Parent.