CS 1110 Review Session
Subclasses and Abstract Classes

Class Invariant
Class invariant is important for the programmer --they can look up everything when necessary and when writing methods. And any reader will benefit as well.

```
/** An instance is a time of day */
public class Time {
   private int hr; // Hour of the day, in range 0..23
   private int min; // minute of the hour, in range 0..59
}
```

What is the class invariant in the above example? Why are they there?

Class invariants are also important in Constructors. How can you tell what the constructors do? For example:

```
/** Constructor: an instance with time t, in minutes, in range
0..24*60-1*/
public Time(int t) {
 hr= t / 60;
 min= t % 60;
}
```

Without seeing the constraint, one would write simply hr = t;

Constructors
The purpose of the constructor is to initialize ALL of the fields of an object so that the class invariant is true. Not just the fields whose values are given as parameters but ALL the fields.

You may also initialize declarations for fields such that you don't need to have an assignment for them in the constructor. For example, in Rhino:

```
int tag= -1; // the Rhino's tag number, >= 0 (use -1 if no tag yet)
```

If a field and the parameter that is to be assigned to it have the same name, then use "this" to refer to the object in which "this appears.

The following won't work because OUR INSIDE-OUT RULE says that both references to radius refer to the parameter:

int radius;

```
 /** An instance with radius radius */
 public C(int radius) {
    radius= radius;
```

```
  }
```

Instead, write it as follows --remember that "this" always refers to the object (i.e. the name of the object) in which it appears.

```
/** An instance with radius radius */
 public C(int radius) {
     this.radius = radius;
 }
```

If YOU do not put in a constructor in a class C, Java inserts this one for you:

```
 public C() {}
```

If you do put in any constructor, Java does NOT add this one.

Important rule
When an object of some subclass SC is created, the inherited fields should be initialized before the fields declared in subclass SC. Java tries to enforce this by requiring that the first statement in ANY constructor that you write a call on another constructor --either a call on a constructor in a superclass:

```
 super(...);
```

or a call on another constructor in this class:

```
 this(...)
```

On the test, you may be asked to write some classes and subclasses. In writing the constructors, remember two things:

(a) Inherited fields are usually private, so they CANNOT be initialized by assigning to them.

(b) It is best to initialize inherited fields using a call on the superclass constructor like mentioned above

Example: Fall 2006 Prelim 3 Question 5
Question 5 (21 points). Consider the classes provided below and answer the following questions.
(a) In class Positive, write the body of the constructor.
(b) In class Rational, write the bodies of the constructor and procedure setPositive. In doing these, keep in mind that the rational number must always be maintained with the denominator > 0
and in lowest possible terms —e.g. the rational number 15/45 is maintained as 1/3 and 5 / –3 as –5/3.
(c) Explain why class Rational overrides procedure setPositive.

```
/** An instance wraps a positive
   integer */
```

```
public class Positive{
   // the positive integer
      private int k;

   /** Constructor: an instance
      with value k.
      Precondition: k > 0 */
      public Positive (int k) {

      }

   /** = this instance's value */
      public int getPositive() {
                return k;
        }

   /** Set the value of this instance
      to n.
      Precondition: n > 0 */
      public void setPositive(int n){
        k= n;
        }
}


/** An instance is a rational number */
public class Rational extends Positive {
  /** The rational number is num / k, where k is the value
     wrapped by the super class.
     Restrictions on fields:
     k is always > 0 and
          num / k is always in lowest possible terms.
     E.g. instead of 10/5 or -5/10, these numbers
     are stored as 2/1 and -1/2.*/

    private int num;

   /** Constructor: an instance with rational number
       num / denom.
       Precondition: denom != 0 */
    public Rational(int num, int denom) {
    }

/** Set the value of the denominator to n.
     Precondition: n > 0 */
    public void setPositive(int n){

        }

  /** Reduce this rational number to the lowest
     possible terms, e.g. 8/24 becomes 1/3 */
    public void reduce(){
```

```
    // YOU DO NOT HAVE TO WRITE THIS BODY
    }
```

Answer:

```
/** Constructor: an instance with value k.
      Precondition: k > 0 */
public Positive (int k) {
    this.k= k;
}

/** Constructor: an instance with rational
       number num / denom.
       Precondition: denom != 0 */
public Rational(int num, int denom) {
    super(Math.abs(denom));
    if (denom < 0)
    this.num= - num;
    else   this.num= num
    reduce();
}

/** Set the value of the denominator to n.
   Precondition: n > 0 */
public void setPositive(int n){
    super.setPositive(n);
    reduce();
}
```

Abstract Classes

(a) Why make a class abstract? So that one can't instantiate it --i.e. create objects of that class.

(b) Why make a method abstract? So that it MUST be overridden in any subclass.

You should know the syntax for abstract classes. Are there any questions about it?