CS100J, Spring 2006 Preparing for Prelim 1: Thursday 23 Feb. 7:30—9:00PM

This handout explains what you have to know for the first prelim.
The website contains the following (you can print them out if you want):

- A file with sample questions and answers.
- A similar prelim from Fall 2003 and Spring 2005, and its answers.

Terms and their meaning

Below, we summarize the terms you should know. You should be able to state the definition of a term like "assignment statement" clearly and precisely. For a Java construct, you should know its syntax. For a Java statement, you should know how to execute it.

• **Expressions**: Types **int**, **double**, **boolean**, **char** (their ranges and basic operations). Casting between types. Narrower type, wider type. You **must** know the methods of class String, as discussed in Lab 04.

• **Variables**: variable, declaration of a variable, assignment statement**.** Four kinds of variables: field, static variables, parameter, and local variables; No where each is declared and what its scope is. See last page of this document.

• **Methods**: Three kinds of methods: procedure, function, constructor. Syntax of a method definition. Parameter of a method. Local variable of a method. Scope of a parameter and a local variable. Be able to write a simple method.

• **Method calls**: How to call each kind of method. Argument of a method call. Restrictions on arguments based on the type of the corresponding parameter. Frame for a method call

• **If-statement and if-else statement.** Their syntax is and how they are executed. If-condition.

• **Block**. What its syntax is and how it is executed.

• **Classes**. What is a class? Class definition. Instance (folder, or object) of a class. The name of a folder. Components: fields and methods. Static and non-static components of a class (where do they go?). What "**this"** means. The new-expression and what it is used for. You should be able evaluate a new-expression by hand, drawing the new folder, filling in the initial values, and yielding the name of the new folder.

• **Subclasses**.How to define a subclass. Inheritance and overriding. Casting with class-types. Narrower and wider class types. You should be able to draw a folder of a class or subclass, given the class definition. What **super** means.

Key concepts

These notes contain short definitions of the basic entities that make up a Java program, along with a description of the Java syntax for them and examples of them. Memorize these definitions. You should know them backward and forward by now, for they form the basis of whatever we do from now on.

On a test, you should be able to write such definitions and examples. What you write must be precise and clear.

Class: A file drawer: contains static components and folders (instances, objects) of that class.

Class definition: a "model", form, or blueprint for the objects (or instances) of the class; a class defines the components of each object of the class. All folders of the class have the same components. Analogy: a blueprint for a house is a design for a house, many houses (objects) can be built from the same blueprint, but they may differ in color of rooms, wallpaper, etc.

Java syntax: **public class** <class name> {
declaration of fields and methods
}

Variable: A named box that can contain a value of some type. For a type like **int**, the value is an integer. For a class-type, the value is the name of (or reference to) an instance of the class —the name that appears on the folder.

Declaration of a variable: a definition of the name of the variable and the type of value it can contain.

Java syntax: <class or type name> <identifier> ;

Examples of variable declarations:

A variable x that can contain an integer: **int** x;
A variable s that can contain the name of an object of class String: String s;
A variable c that can contain a boolean value: **boolean** b;

Method: A parameterized sequence of statements, whose execution performs some task. We have three kinds of method: procedure, function, constructor.

A method should be accompanied by a comment that says what the method does. This is the *specification* of the method. The comment has to be precise and clear. A potential user of the method should be able to look only at the comment and the list of parameters to know how to call the method; they should not have to look at the body of the method.

Example. When you want to bake a cake, you look at the title of a recipe, a short description, and the list of ingredients to determine whether you want to use that recipe —not the list of instructions to bake it.

A procedure is a method that performs some task (and doesn't return a value)

Java syntax:

```
/** Comment that explains what the method does */
public void <method name> (<parameters>) {
    Sequence of statements to execute
}
```

Example:

```
/** Raise the salary by n dollars if the salary is < $20000 */
public void raiseSal(double n) {
   if (salary < 20000)
       salary= salary + n;
}
```

Example procedure call: raiseSal(20*y);

A function is a method that performs some task and returns a value. Instead of keyword **void**, the type of the returned value is used. Statement **return** <value>; is used to terminate execution of a function call and return <value>.

Java syntax:

```
/** Comment that explains what the function does. It should include something like "= …" to
          describe what the function value is. */
public void <method name> ( <parameters> ) {
    Sequence of statements to execute
}
```

Example:

```
/** Yield the maximum of x and y */
public int max (int x, int y) {
   if (x>= y) return x;
      return y;
}
```

Example of a function call of max (within some statement): z= 1 + max(x,y);

A constructor is a method that initializes (some of) the fields of a newly created object.

Java syntax:

```
/** Constructor: an instance that …(describe initial values of fields). */
public <class name> ( <parameters> ) {
    Sequence of statements to execute
}
```

Example:

```
/** Constructor: an instance with title t and chapter number n */
public Chapter (String t, int n) {
```

```
    title= t;    chapterNumber= n;
}
```

Example of a constructor call (only within a new-expression!): **new** Chapter("tt", 5)

You MUST know how to evaluate a new-expression **new** C(…):

1. Create a new instance of class C and store it in C's file drawer
2. Execute the constructor call C(…)
3. Use the name of the newly created instance as the value of the new-expression.

Execution of an assignment statement stores a value in a variable.

Java syntax: <variable name> = <expression> ;

Restriction: The type of the expression cannot be narrower than the type of the <variable name>

Examples:
```
b= 2+c;
s= "Cardie" + "  " + yearHired";
```

Please, always put no blanks before = and one blank after =, to make it look unsymmetric and remind you that it is not an equality test but an assignment.

A block is used to unify a sequence of statement into a single statement.

Java syntax: { sequence of statements }

Example: Here is a sequence of two statements:
```
a= 10;
if (a < c) then
    a= c;
```
Here is a single statement, which is a block
```
{  a= 10;
   if (a < c) then
       a= c;
}
```

Execution of a conditional statement allows a choice of execution.

Java syntax:
```
if  ( <boolean expression> )
        <statement>
```
or
```
if  ( <boolean expression> )
      <statement 1>
else <statement 2>
```

The first form is executed as follows: if <boolean expression> is true, then execute <statement>

The second form is executed as follows: if <boolean expression> is true, then execute <statement 1>;
if the <boolean expression is false, then execute <statement 2>.

A subclass B (say) is a class that extends another class A (say). This means that an instance of B has all the fields and methods that an instance of A has, in addition to the ones declared in B.

Java syntax:
```
public class <class name> extends <class name> {
 declarations of fields and methods
}
```

Access modifiers. Suppose d is an instance of Employee, where class Employee is declared as:

```
public class Employee {
  <access modifier> int x;
```

```
…
}
```

If the <access modifier> is: **public**, then field d.x can be referenced anywhere that d can be referenced.
If **private**, then field d.x can be referenced anywhere within class Employee that d can be referenced

kinds of variables: local variables, parameters, and fields (non-static)

```
public class Class1      {
public int x;
        public int y;                          //  x is a field of instance variable. It appears in every folder

        public voidClass1 (int z)              // z is a parameter.
                {z= z; y= 2*z;}

        // Set y to the maximum of p and –p
        public void sety(int p) {              // p  is a parameter
           int x;                              // x is a local variable of method sety. It cannot be used
           x= p;                               // outside the method. It is local to the method.
           if (p > –p)
                   x= –p;
                   y= p;
           }
```

The scope of a name is the set of places in which it can be referenced.

A variable can be declared only once within a method. Such a variable is sometimes called a local variable (of the method).

The scope of a local variable of a method is the sequence of statements following it.

```
        Example:
        /** specification of method */
        public test(int p) {
                y= p;
                int x;          // The scope of x starts at the next statement and goes
                x= p;           // to the end of the block in which the declaration of x appears
                if (p > –p)
                   x= –p;
                y= p;
        }
```

The scope of a parameter of a method is the method body.

```
        Example:
        public test(int p) {                   // The scope of parameter p is the method body
                if (y= p); {
                        int x;                 // The scope of x starts at the next statement and
                        x= p;                  // goes until the end of the block in which the declaration
                        if (p > –p)            // of x appears. It does not include the last statement y= p.
                                x= –p;
                        }
                y= p;
}
```

The scope of a field of a class consists of:

(1) the bodies of all methods declared in the class and
(2) all declarations of fields that follow the declaration of the field.