CS100J About Prelim II (Thursday evening, 17 March, 7:30--9:00PM)

You should know everything that you needed to know for the first test --we review this material below. The new material since prelim 1 consists of these topics. You should know this material thoroughly:

1. While loops. Know how to execute a while loop. We will give you a precondition, postcondition, and loop invariant, and you will have to develop the loop plus initialization from it. Be able to work with assertions and loop invariants, as done in class since 3 March. p. 233--253.

2. Apparent and real types, casting, operator instanceof, and function equals. Study pp. 148--155. Know how to write a function equals. See the discussion at the very end of this document!

3. Class Vector. Be able to use class Vector. On any question about Vector, we will specify any methods of class Vector that you need to answer the question --you don't have to memorize them.

4. Ability to write functions (as we have been doing in several labs, now).

Definitions. Know the following terms, backward and forward. Wishywashy definitions will not get much credit. Learn these not by reading but by practicing writing them down, or have a friend ask you these and repeat them out loud. You should be able to write programs that use the concepts defined below, and you should be able to draw folders and frames for calls.

Variable: a name with associated value OR A named box that can contain a value of some type or class. For a type like **int**, the value is an integer. For a class, it 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 or class of value it can contain. Basic syntax: *type variable-name* ;

Four kinds of variables: parameter, local variable, instance variable(or field), static variable (or class variables).

Parameter: A variable that is declared within the parentheses of a method header. The variable is drawn in a frame for a call on the method --at the time the frame is created.

Local variable: A variable that is declared in the body of the method. The variable is drawn in a frame for a call on the method --at the time the frame is created.

Instance variable: A variable that is declared in a class without modifier static. An instance variable is drawn in every folder of the class.

Static variable: A variable that is declared in a class with modifier static. An static variable is placed in the file drawer for the class in which it is declared --when program execution starts.

Three kinds of methods: procedure, function, constructor:

A procedure definition has keyword **void** before the procedure name. A procedure call is a statement.

A function definition has the result type in place of void. A function call is an expression, whose value is the value returned by the function.

A constructor definition has neither keyword **void** nor a type, and its name is the same as the name of the class in which it appears. The constructor call is a statement, whose purpose is to initialize (some of) the fields of a newly created folder.

Argument: An expression that occurs within the parentheses of a method call (arguments are speareted by commas).

Folder (manila folder, object, or instance) of a class. An entity that is drawn like a manila folder. It has a name or label on its tab. Its contents are the instance methods and instance fields defined in the class definition.

New-expression. An expression of the form **new** class-name (arguments). It is evaluated as follows: (1) create a new folder of class class-name and put it in class-name's file drawer. (2) Execute the constructor call class-name (arguments); where the method called is one that appears in the newly created folder. (3) Yield as the result of the new-expression the name of the folder created in step (1).

Frame for a method call. The frame for a method call contains: (1) the name of the method and the program counter, in a box in the upper left, (2) the scope box (see below), (3) the local variables of the method, (4) the parameters of the method.

The scope box for a call contains: For a static method: the name of the class in which the method appears. For an instance method: the name of the folder in which the instance appears.

To execute a method call:

1. Draw a frame for the call
2. Assign the (values of) the arguments to the parameters..
3. Execute the method body. When a name is used, look for it in the frame for the call. If it is not there, look in the place given by the scope box.
4. Erase the frame for the call.

Folder. We assume you can draw a folder, or instance of a class. For subclasses, remember that the folder has more than one partition. Look at the homework we had on drawing folders.

Class Object. Every class that does not explicitly extend another subclass automatically extends class Object. Class Object has at least two instance methods: toString and equals.

Calling one constructor from another. In one constructor, the first statement can be a call on another constructor in the same class (use keyword **this** instead of the class-name) or a call on a constructor of the superclass (use keyword **super** instead of the class-name).

Inheriting methods and fields. A subclass inherits all the components (fields and methods) of its superclass.

Overriding a method. In a subclass, one can redefine a method that was defined in a superclass. This is called overriding the method. In general, the overriding method is called. To call the overriden methodm (say) of the superclass, use the notation **super**.m(...) --this can only be done in methods of the subclass.

Real and apparent class. A variable x defined using, say, CLAS x; has apparent class CLAS. The apparent class is used in determine whether a reference to a field or method is syntacticly legal or not. One can write x.m(...), for example, if and only if method m is declared or is referenceable in class CLAS. The real class of x is the class of an object that is in x. It could be a subclass. If x.m(...) is legal, then it calls the method that is accessible in the real class, not the apparent class. P. 148–154.

Casting. Just as one can cast an **int** i to another type, using, say, (**byte**) i or (**double**) i, one can cast a variable of some class-type variable to a superclass or subclass. Look in PLive to see about this. See p. 152–154.

Operator instanceof. ob **instanceof** C has the value of "object ob is an instance of class C". p. 152--153.

Function equals(Object ob). Suppose this instance function occurs in class C. This boolean function returns the value of "ob is not null, is the name of an object of class C, and is equal to this object". What "equal to this object" means depends on the writer of the method and should be specified in a comment before the function. In class Object, it means "this object and ob are the same object". Generally, when one writes such an equals(Object ob) function, "equal to this object" means "the fields of this object and object ob are equal".See p. 154. Method equals on p. 154 is written incorrectly, because e has to be cast to Employee in order to reference the fields. It should be:

```
/** = "e is an Employee, with the same fields as this Employee */
public boolean equals(Object e) {
    if (e != null) return false;
    if (!(e instanceof Employee)) return false;
    Employee ec= (Employee) e;
    return name == ec.name && start == ec.start && salary == ec.salary;
}
```