CS1110 About Prelim II (Thursday evening, 12 March, 7:30--9:00PM, Uris Hall G01)

Review session, 1-3PM, Sun, 8 March, Phillips 101.

You should know everything that you needed to know for the first test —we review this material beginning in the next column. The new material since prelim 1 consists of the following topics. You should know this material thoroughly —note that loops will *not* be covered on this exam.

1. Apparent and real types, casting, operator instanceof, and function equals. Study pp. 148–155. Know how to write a function equals. See the discussion below under "Equals function".

2. 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.

3. Knowledge of class String. Know about the basic String functions length(), charAt(k), substring(k, j), and substring(k). We will specify any others that you need.

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

5. Recursive functions. Be able to write recursive functions, as we have done in class and lab. Know the important points: (1) precise specification, (2) base case(s), (3) recursive case(s), and (4) progress toward termination.

6. Abstract classes and methods. Know the purpose of an abstract class (make a class abstract so that it cannot be instantiated) and how to make a class abstract. Know the purpose of an abstract method (make a method abstract so that it must be overridden in any non-abstract subclass) and how to write an abstract method. See pp. 163–164.

7. Style. We have made comments from time to time on programming style —using good specifications, class invariants, indenting, etc. All the material on style is presented in Chapter 13 of the class text. You should be familiar with this material, so read it for the prelim. You will be especially responsible for Sections 13.3.1–2 on specifications and Section 13.4 on describing variables.

Concepts and Definitions

Know the following terms, backward and forward. Wishy-washy 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 objects and frames for calls.

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

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.

Calling one constructor from another. In one constructor, the first statement must 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). If not, the constructor **super**() {} is inserted by Java.

equals(Object ob). Suppose this instance function is declared in class C. This boolean function returns the value of

> "ob is the name of an object of class C and is equal to this object".

The meaning of "equal to this object" 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". Thus, for the equals defined in Object, b.equals(d) and b == d have the same value (except in the case b = **null**!).

Generally, "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 instanceof Employee))
      return false;
   Employee ec= (Employee) e;
   return name.equals(ec.name) &&
      start == ec.start &&
      salary == ec.salary;
}
```

Folder. We assume you can draw a folder, or object 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.

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.

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

Inside-out and bottom-up rules. Used in determining which declaration a variable reference or function call refers to. Look them up in the text.

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

Method call execution or evaluation: (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.

Methods, kinds: 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 the fields of a newly created folder.

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

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.

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 overridden method m (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 declared using, say, CLAS x; has apparent class CLAS. The apparent class is used in determining whether a reference to a field or method is syntactically legal. One can write x.m(...), for example, if and only if method m is declared in or inherited by 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. pp. 148–154.

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 object in which the instance appears.

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 tab of the object.

A variable declaration has the basic syntax: *"type variable-name* ;". Its purpose is to indicate that a variable of the given type is to be used in the program

There are 4 kinds of variable: parameter, local variable, instance variable (or field), static variable (or class variable).

A local variable is declared in the body of the method. The variable is drawn in a frame for a call on the method —when the frame is created.

An instance variable is declared in a class without modifier **static**. An instance variable is drawn in every folder of the class.

A parameter is declared within the parentheses of a method header. The variable is drawn in a frame for a call on the method —when the frame is created.

A static variable is declared in a class with modifier **static**. A static variable is placed in the file drawer for the class in which it is declared —when program execution starts.

Wrapper class. With each primitive type (e.g. **int**) there is an associated *wrapper class* (e.g. Integer). The wrapper class serves two purposes: (1) wrap or contain one value of the primitive type, so the primitive-type value can be treated as an object. (2) House some useful methods for operating on values of the primitive type.