CS1110, Spring 2011. Preparing for Prelim 1.
Tuesday, 8 March, 7:30–9:00PM, Uris Hall G01
Review session: Sunday, 21 Feb. 1:00-3:00PM, Phillips 101

This handout explains what you have to know for the first prelim. The website contains several previous CS1110 prelims and a file with sample questions and answers. To prepare for the prelim, you can (1) practice writing programs/methods in DrJava, (1) ***read the text***, (2) memorize definitions, principles, strategies for programming.

The prelim will *not* cover recursion. It covers material up to and including material in lecture on 23 Feb.

Terms and their meanings

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

- **Expressions**: Types **int**, **double**, **boolean**, **char** (their ranges and basic operations). Casting between types. Narrower type, wider type. Know how to use the conditional expression `(<bool exp> ? exp1 : exp2)` .

- **Variables**: variable, declaration of a variable, assignment statement**.** Four kinds of variable: field, static variable, parameter, and local variable; know where each is declared and what its scope is. See last page of this document.

- **Methods**: Three kinds of method: 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. Be able to execute a method call using the 4 steps: draw the frame, store argument values in the parameters, execute the method body, erase the frame.

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

- **Block**. Its syntax and how it is executed. It is just a statement of the form "`{ … }`".

- **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?). The new-expression and what it is used for. What **`this`** means: in a method, **`this`** evaluates to the object in which the method occurs. What **`super`** means: in a method, **`super`** evaluates to the object in which the method occurs but only starting at the partition above the one in which **`super`** occurs. B able evaluate a new-expression by hand: draw a new object, execute the constructor call, and yield the name of the new object as the value of the expression.

- **Subclasses**. How to define a subclass. Inheritance and overriding. Constructors in a subclass. If you don't put in a constructor, Java puts this one in: **`public`** <class-name>`(){}` . The first statement must be a call **`super`**`(…);` on a constructor of the superclass or a call **`this`**`(…)` on another constructor in this class. If there is none, **`super`**`();` is used. You should be able to draw an object of a class or subclass, given the class definition.

- Class `String`. Know these basic methods of class `String`, as discussed in Lab 04: `length()`, `charAt(i)`, `substring(i)`, and `substring(i,j)`. You will be asked to write a function that manipulates strings. We will define all methods of class `String` that you need except the ones mentioned above. Sec. 5.2.

- Class `Vector`. Now how to create and use a `Vector`. We will define all methods of class `Vector` that you need. Sec. 5.3

- Wrapper classes. A primitive type (**e.g. int** has a corresponding wrapper class (e.g. Integer), each object of which contains or "wraps" one value of that type. Know the two reasons for having the wrapper class. Chap. 5.

Key concepts

Below are short definitions of the basic Java entities, along with a description of the Java syntax and examples of them. Memorize the definitions. Know them backward and forward, for they form the basis of whatever we do. On a test, you should be able to write such definitions and examples. What you write must be precise and clear.

<u>Class</u>: A file drawer: contains static components and folders (instances, objects) of the class.

<u>Class definition</u>: 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 (objects) 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.

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

Different kinds of variables require slightly different declarations. For example, declarations of local variables end in ";", declarations of parameters are separated by ",", and declarations of fields have an access modifier **`private`** or **`public`** and end in ";".

Examples of variable declarations:
A local variable `x` that can contain an integer: **`int`** `x;`
A local variable `s` that can contain the name of an object of class `String`: `String s;`
A field `b` that can contain a boolean value: **`private boolean`** `b;`

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

A method should be accompanied by a javadoc 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 at the list of instructions to bake it.

Procedure: 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 < $20,000 */
public void raiseSal(double n) {
    if (salary < 20000)
         salary= salary + n;
}
```

Example procedure call: `raiseSal(20*y);`

Function: a method that performs some task and returns a value. Instead of **`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:
```
/** = 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);`

Constructor: 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 (draw) 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: evaluate the expression and store its value in the 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";`

Always put no blank 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 statements into a single statement.

Java syntax: `{` <sequence of statements> `}`

Example: A sequence of two statements:

```
a= 10;
if (a < c) then
        a= c;
```

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, execute <statement>.

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

A subclass `D` (say) is a class that extends another class `C` (say). This means that an instance of `D` inherits (has) all the fields and methods that an instance of `C` has, in addition to the ones declared in `D`. Know how to draw objects of a subclass. Know that every class that does not explicitly extend another class automatically extends class `Object` and that `Object` contains at least two functions: `equals(Object)` and `toString()`.

Java syntax:

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

Access modifiers. Suppose `d` is an instance (or an object) of `C`, where class `C` is declared as:

```
public class C {
```

```
          <access modifier> int d;
          ...
            }
```

If the <access modifier> is **`public`**, then field `d.x` can be referenced anywhere that `d` can be referenced.
If it is **`private`**, then field `d.x` can be referenced only in methods in class `C`.

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

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

     public void Class1(int z)              // z is a parameter.
             { 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= x;
     }
```

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

A variable declared within a method is called a local variable (of the method). Its scope is the sequence of statements following it (within the containing block).

```
     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 the bodies of all methods declared in the class and all declarations of fields that follow the declaration of the field