CS102

Introduction to
data structures, algorithms, and
object-oriented programming

DAY 5

1

switch

A switch statement allows you to test the value of an expression and, depending on that value, to jump directly to some location within the switch statement. You should always use break or return statements in a switch.

The value of the expression can be one of the primitive integer types int, short, or byte. It can also be the primitive char type or it can be a String.

The expression **cannot** be a double or float value

2

switch example

```
switch ( grade ) {// (Assume grade is a char variable or expression.)
         case 'A':
            System.out.println("Great job!");
            break;
         case 'B':
            System.out.println("Good!");
            break;
         case 'C':
               System.out.println("Not bad");
               break;
         case 'D':
            System.out.println("Could be better.");
            break;
         case 'F':
            System.out.println("You need a break.");
            break;
         default:
            System.out.println(grade + " is not a valid grade.");
            break;

}
```

3

arrays § 3.8

A data structure in which the items are arranged as a numbered sequence, so that each individual item can be referred to by its position number.

All the items in an array must be of the same type, and the numbering always starts at zero. An array is a list of variables, each accessible by the array name and position number of the variable.

An array is, technically, an object, so the process of creating one requires an instantiation with the keyword new.

4

arrays (cont.)

An array can be of any type and must first be declared:

```
String[] name;     // declaration of String array
int[] age;         // declaration of int array
boolean[] leftHanded; // declaration of boolean array
```

Then the array must be instantiated:

```
name = new String[1000];  // each with initial value null
age = new int[5];       // each with initial value 0
leftHanded  = new boolean[100]; // each is false init
```

After instantiation, the specified number of boxes will be created in memory and reserved for that type.

5

arrays (cont.)

To put values into the array, you use the array name and position number to store a value at that position:

```
name[5] = “Matilda”;
```

The length of a array is stored with the array as a field name accessible as, for example   name.length // notice these are
                                                                        age.length   // NOT method calls

Having access to the length of every array allows them to be easily traversed with a for loop to go through each element:

```
// this for loop prints out all the elements in array age
for (int i = 0; i < age.length; i++) {
   System.out.println( age[i] );
} // end for
```

6

2-dimensional arrays

Declaration and instantiation example:

int[][] matrix = new matrix[10][5];

This line would create a matrix with 10 rows and 5 columns, initially all 0. You would need to add values for each of the 50 ints in the array

Often initialized or printed in nested for loops:

```
int row, col; // loop-control-variables
for ( row = 0; row < 5; row++ ) {
    for ( col = 0; col < 7; col++ ) {
        System.out.printf( "%7d", A[row][col] );
    } // end inner for
    System.out.println();
} // end outer for
```

random numbers

The random method is a static member of the Math class. The call Math.random() produces a double between 0.0 and 1.0, inclusive. To use the Math.random() function to get a number between 1 and 10, you would use the following call:

int rNum = (int)(Math.random() * 10) + 1

The (int) operator truncates the real number to produce an int. This type of operator is called a "cast".

Static Methods

In the Eck book, a *function* is a method whose job is to compute and return some value. The return-type is used to specify the type of value that is returned by the function.

Static vs non-static methods:

In a running program, a *static method* is a member of the class that contains it. Called on the class name (using the . operator after the class name).

A *non-static method* can be called only on objects of the class type and the methods are members or fields of the object.

Static Methods

A program can contain many methods, but only one main method.

All methods are contained with a class block and no method can be written inside another method (although methods can call each other). Methods can contain any kind of statement (except package and import statements.)

General form of method headers:

```
modifiers return-type methodName ( ParType parName) {
{
    statements
}
```

Parameter lists

Parameters are part of the interface of a method. They contain information that is used inside when the method is called.

Parameters are not given values until the method is called and arguments are passed into the method.

Unlike variables created outside a method, the variables contained within the parameter list must each have its type specified and each TypeName varName must be separated by commas.

Calling Methods

The syntax for calling any method that exists inside the same class as the method that is calling it from a non-static context is as follows:

methodName(argument(s));

The syntax for calling a static method in the same or another class is:

ClassName.methodName(argument(s));

If the method returns a value, you should declare a variable to hold that value, use it as an argument to a method, or use it as part of an expression.

General Rule of Java Program Structure

Methods are never written inside methods (unlike the local special form in Racket). But methods can call other methods.

Methods can have any number of parameters, including 0.

The first line of a method the method header or signature, e.g.

modifiers return-type subroutine-name (parameter-list)

13

General Rule of Java Program Structure

The method and the parameter list is called the *method signature*. As we discussed in the last class, we can OVERLOAD methods by writing many methods with the same name in a class, but each must have a unique parameter list.

subroutine-name (parameter-list)

14

Member Variables

Local variables are those declared inside a method.

Global variables are declared inside a class and are called member variables or fields.

Member variables can be static, meaning they are used in expressions following the class name.

Member variables are assigned a default value.

Local variables are not assigned a default value and must be initialized before being used in an expression.

15

Returning values from a Method

The author of our text calls methods that return a value *functions*. These methods must have a return type that is not void.

You need to explicitly return a value of the type given in the method header line.

return expression;

Only one value can be returned in a return statement. Executing a return statement breaks out of the method and returns control to the statement that called the method.

You can include more than one return statement inside a method, but there must be a return statement on each branch of execution.

The return type must match the type given in the method header.

16

Method Contracts

Prior to writing each method, you should write a comment, or contract, to explain what the method does and any assumptions about parameters.

Parameters can be of any type, including arrays.

```
String[] names ={Gerry, Roger, Helen, Ann};
countLetters(names);

static int countLetters(String[] noms) {
   int count = 0;
   for (int i = 0; i < noms.length; i++) {
      <add code here to count letters>
      <add statement here to return count>
   }
}
```

17

Write a method to return the reverse of a String

Since Strings are numbered like arrays, and because the String class has a function length(), it is easy to use a for loop to build and return the String in reverse.

(in class exercise)

Convert the method to a palindrome checker.

18