CS102
Introduction to
data structures, algorithms, and
object-oriented programming

DAY 6

1

Summary of 2/13/17

Objects encapsulate data (instance variables) and behaviors (instance methods).

The non-static portion of classes describe objects.

Object classes are used to create objects; they are a blue-print for objects; they have few or no static members.

The User class below would set the same data in all objects created from it.

```
class User {
   public static String name;
   public static int age;
}
```

2

Object class example

```
class PlayerData {
   static int playerCount = 0;
   String name;
   int age;

   PlayerData(String name, int age) {
      this.name = name;
      this.age  = age;
      playerCount++;
   }

   public static void main(String[] args) {
      PlayerData p1 = new PlayerData("Mac", 24);
      PlayerData p2 = new PlayerData("Cam", 25);
   }
}
```

p1 and p2 access same copy of playerCount. They each have their own name and age fields.

3

Very Important Points

1. Declaring a variable of object type does not create an object.
2. In Java, no variable can ever hold an object. A variable can contain only a reference (pointer or memory address) to that object.
3. Java arguments are "pass by value".

   For primitive type variables, the value is just passed as an argument to the method, the value is not changed in the calling program.

   For object type variables passed into methods, the address of the object in memory is passed, so the object's internal state can be permanently changed within a method (arrays are included as objects).

4

.equals and .toString methods

String ethel = "Ethel";
String fred = "Fred";

To compare these String object, you can't do the following:

```
if (ethel == fred) {...}
```

Objects must be compared using the .equals method (defined for most built-in classes):

```
if (ethel.equals(fred)) {...}
```

Methods that every new object class should contain:

```
public boolean equals(Student other) {
    if (this.name.equals(other.name) &&
        this.age == other.age) {...}}

public String toString() {
    return "Name is "+this.name+", age is "+this.age;
}
```

5

.equals method

.equals returns true if the argument pass to it is not equal to null and is of the same type and has same field values.

```
Integer x = 5, y = 10, z = 5;
Short a = 5;

System.out.println(x.equals(y));   //?
System.out.println(x.equals(z));   //?
System.out.println(x.equals(a));   //?
```

6

.toString method

.toString is used to return the contents of an object's fields in a String.

It is convenient because if an object does have a toString method, there is no need to actually call toString inside a print statement.

7

arrays § 3.8

A data structure in which the items are arranged as a 0-based numeric 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.

8

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.

Declaration and instantiation can occur on the same line.

9

arrays (cont.)

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

The new keyword is only used to create new objects and, since it is used to create an array, it follows that an array is an object.

10

arrays (cont.)

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

```
name[5] = "Penny";
```

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
```

11

arrays (cont.)

Arrays are generally processed using for loops. The loop knows when all the elements are processed because of the length field stored with each array object.

Eg, Suppose we have an array of Strings called nameList that contains 10 String objects in positions 0...9. Then the following for loops are identical in action:

```
for( int i = 0; i < nameList.length; i++) { // for loop
   System.out.println( nameList[i] );
}
```

```
for ( String  name  :  nameList) {   // for-each loop
     System.out.println( name );
}
```

12

arrays (cont.)

Another way to declare and instantiate an array, if you know what the values in the array will be, is to enumerate the list of values in the same statement as the declaration:

```
int[] squares = {1, 4, 9, 16, 25, 36, 49};
```

This enumeration of values can occur only on the same line as a declaration statement.

13

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 printed in nested for loops:

```
int row, col; // loop-control-variables
for ( row = 0; row < 10; row++ ) {
    for ( col = 0; col < 5; col++ ) {
        System.out.printf( "%7d", matrix[row][col] );
    } // end inner for
    System.out.println();
} // end outer for
```

14

arrays (cont.)

You can also create an enumerated 2D array:

```
int[][] squares = {{1, 4, 9},
                   {16, 25, 36},
                   {49, 64, 81}};
```

Here, the references
squares[0][0] is 1

We can print all the elements of a doubly-nested for loop as shown below:

```
for ( int i = 0; i < squares.length; i++) {
    for (int j = 0; j < squares[i].length; j++) {
        System.out.print(squares[i][j]);
    }
}
```

15

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. A static method is called on the class name.

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

16

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
}
```

17

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.

18

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.

19

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)

20

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)

21

Variable types

Local variables are those declared inside a method.

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

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

Global variables are assigned a default value.

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

22

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.

23

Method Contracts

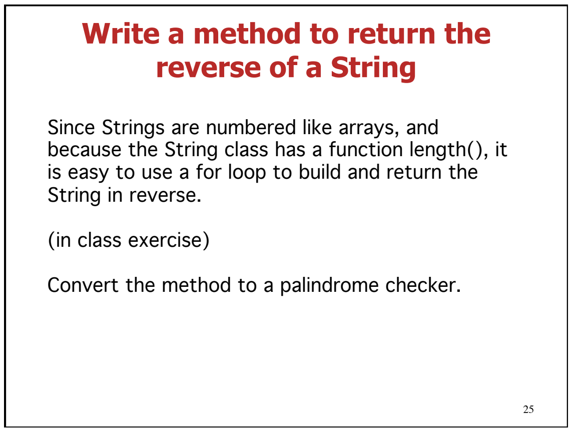
Prior to writing each method, you should write a comment, or contract, to explain what the method does and any assumptions about parameters.

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>
   }
}
```

24



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.

25