Chapter 2: Java Review

Java Review

To start with, here's an overview list of the Java topics covered in my first course. You should already be familiar with many of them.

- Classes and objects, and how they're related
- Class members, both data elements and methods
- Methods and constructors
- Primitive data types
- Parameter passing
- Variables, constants, and literals
- Console input and output
- Arithmetic expressions
- Logical expressions
- Making decisions: if-else and switch-case statements
- Repeating actions: while and do-while loops
- Arrays
- Creating and drawing in a window

Whew! That's quite a list! We're going to do a little more in-depth review of these topics in this lesson, but if you're unfamiliar with many of these terms and concepts, and the review doesn't refresh your memory, you might want to consider taking the first course before taking this one.

If you do recognize all these, that's great! Let's go over what they mean so we're ready to tackle the new Java concepts in this course.

Classes and Objects

A *class* describes some entity (thing) that we want to represent in a computer. For example, if we were writing software to support a school environment, we would need to represent the entities involved in the processes in our software. What entities would we need to represent in a school? Well, there are students, teachers, classrooms, schedules, grades, classes, and so on. Note that an entity doesn't have to be a physical thing. It can be an idea or concept, like the schedules and grades I mentioned.

When I define a class in Java, I have to tell the program what data to store about the entity, such as a student's name, address, or phone number. I also have to tell Java what the entity can do in my system. A student can register for a class, turn in assignments, get grades, and so on.

We declare data elements (or attributes) as *variables* in the class definition. And we declare the actions of a class as its *methods*. More about them in a minute.

Once we've declared a class and defined its data and actions, we can use it to create *objects*. Objects are individual entities of the types defined by the class. For example, in the school environment, we could create Student and Teacher classes. From those, we could create Student objects to represent the students Tom, Jane, and Bill. We could also create Teacher objects to represent the teachers Ms. Carpenter, Mr. Jones, and Ms. Smith.

These objects could then interact using their methods—to record, for example, that Tom is in Ms. Carpenter's class, Jane is in Ms. Smith's class, and Bill is in Mr. Jones' class.

Class Members

Classes have *data members* (also called attributes and fields) and *method members* (also called actions). Data members are normally *instance variables* in the class, or variables that are part of each instance (or object) created from the class. For example, each Student object in the example we talked about earlier would have variables in which to store a student's name, address, ID number, gender, birthday, and so on. These declarations might take this form:

```
public class Student {

    private String firstName;

    private String lastName;

    private String address;

    private int studentID;

    private Date birthDate;

    private char gender;

    . . . // other declarations and methods not shown

}
```

We usually declare instance variables as *private* so that no other program can manipulate them directly. Methods normally handle setting and retrieving the values in these fields.

Every Student object that we create from this class will have these fields, and each object can have its own values in the fields.

Methods and Constructors

Methods are members of the class, too. We declare these as well, usually after the variables, and most of the time we declare them as *public* so other programs can call them up. Method declarations also define what type of value the method will return (if any) and what information to expect as arguments when a program calls the method. We call that piece of the declaration the method's *signature*.

Method names are normally verbs that describe what the methods do. By tradition, their names start with lowercase letters, and each word in the name after the first word starts with a capital letter. For example, *setFirstName()* and *getGender()* are conventional method names.

Besides its signature, we also have to define a method's actions, and we do that in the *body* of the method, between the curly brackets ({}). For example, we could define methods in the Student class to manage the birthDate field like this:

```
public class Student {

     . . . // variable declarations and some methods are not shown

    public void setBirthDate(int month, int day, int year) {

        birthDate = new Date(year, month, day);

    }



    public Date getBirthDate(){

        return birthDate;

    }

}
```

Constructors are special methods that we can use to construct, or initialize, new objects of the class. You can identify constructors easily; they always have the same name as the class and never have a return type. Java calls a constructor whenever a program creates an object from a class. For example, a constructor for the above Student class could look like this:

```
public Student() {

    firstName = "";

    lastName = "";

    address = "";

    studentID = 0;

    birthDate = new Date();

    . . .

}
```

The last line above, which initializes the variable birthDate, also invokes another constructor, from the Date class, with the code `new Date()`.

Primitive Data Types

Most data types we use in Java are classes, but Java does have some primitive types that we use to build all the other classes. Here are the numeric types:

Java's Numeric Types

Type	Size	Minimum Value	Maximum Value
`byte`	1 byte (8 bits)	-128	127
`short`	2 bytes (16 bits)	-32,768	32,767
`int`	4 bytes (32 bits)	-2,147,483,648	2,147,483,647
`long`	8 bytes (64 bits)	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
`float`	4 bytes (32 bits)	$\pm 1.4 \times 10^{-45}$	$\pm 3.4028235 \times 10^{38}$
`double`	8 bytes (64 bits)	$\pm 4.9 \times 10^{-324}$	$\pm 1.7976931348623157 \times 10^{308}$

In addition to the numeric types, Java also has two other primitive types: one for characters and one for Boolean values. A variable of the *char* type can hold a single character, such as *A*, *Z*, *!*, *9*, and so on. A *boolean* variable holds one of two values: It must be either true or false.

Those are all the basic types that Java has. All other types are classes, including any that we write, and they're all made from the eight primitive types. They may include other classes, too; but ultimately, all classes trace their data types back to these primitives.

I hope all your Java knowledge is starting to come back to you by now! We still have a lot to review, so let's keep going.