National University

Faculty of Computer Since and Technology

Object Oriented Programming Lec (6)

Introduction to Methods

Declaring a Method with a Parameter

- Car analogy
- Pressing a car's gas pedal sends a message to the car to perform a task—make the car go faster.
- The farther down you press the pedal, the faster the car accelerates.
- Message to the car includes the task to perform and additional information that helps the car perform the task.
- Parameter: Additional information a method needs to perform its task.

- A method can require one or more parameters that represent additional information it needs to perform its task.
- Defined in a comma-separated parameter list
- Located in the parentheses that follow the method name
- Each parameter must specify a type and an identifier.
- A method call supplies values—called arguments—for each of the method's parameters.

```
1  // Fig. 3.4: GradeBook.java
2  // Class declaration with one method that has a parameter.
3
4  public class GradeBook
5  {
6     // display a welcome message to the GradeBook user
7     public void displayMessage( String courseName )
8     {
9        System.out.printf( "Welcome to the grade book for\n%s!\n",
10          courseName );
11    } // end method displayMessage
12 } // end class GradeBook
```

Fig. 3.4 | Class declaration with one method that has a parameter.

```java
// Fig. 3.5: GradeBookTest.java
// Create GradeBook object and pass a String to
// its displayMessage method.
import java.util.Scanner; // program uses Scanner

public class GradeBookTest
{
   // main method begins program execution
   public static void main( String[] args )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      // create a GradeBook object and assign it to myGradeBook
      GradeBook myGradeBook = new GradeBook();

      // prompt for and input course name
      System.out.println( "Please enter the course name:" );
      String nameOfCourse = input.nextLine(); // read a line of text
      System.out.println(); // outputs a blank line

      // call myGradeBook's displayMessage method
      // and pass nameOfCourse as an argument
      myGradeBook.displayMessage( nameOfCourse );
   } // end main
} // end class GradeBookTest
```

```
Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

➢ Scanner method nextLine

- Reads characters typed by the user until the newline character is encountered
- Returns a String containing the characters up to, but not including, the newline
- Press Enter to submit the string to the program.
- Pressing Enter inserts a newline character at the end of the characters the user typed.
- The newline character is discarded by nextLine.

More on Arguments and Parameters

- The number of arguments in a method call must match the number of parameters in the parameter list of the method's declaration.
- The argument types in the method call must be "consistent with" the types of the corresponding parameters in the method's declaration.

Instance Variables, *set* Methods and *get* Methods

➢ Local variables

- Variables declared in the body of a particular method.
- When a method terminates, the values of its local variables are lost.
- Recall that an object has attributes that are carried with the object as it's used in a program. Such attributes exist before a method is called on an object and after the method completes execution.

- A class normally consists of one or more methods that manipulate the attributes that belong to a particular object of the class.
  - Attributes are represented as variables in a class declaration.
  - Called fields.
  - Declared inside a class declaration but outside the bodies of the class's method declarations.

➢ Instance variable

- When each object of a class maintains its own copy of an attribute, the field is an instance variable
- Each object (instance) of the class has a separate instance of the variable in memory.

```
1  // Fig. 3.7: GradeBook.java
2  // GradeBook class that contains a courseName instance variable
3  // and methods to set and get its value.
4
5  public class GradeBook
6  {
7     private String courseName; // course name for this GradeBook
8
9     // method to set the course name
10    public void setCourseName( String name )
11    {
12       courseName = name; // store the course name
13    } // end method setCourseName
14
15    // method to retrieve the course name
16    public String getCourseName()
17    {
18       return courseName;
19    } // end method getCourseName
20
21    // display a welcome message to the GradeBook user
22    public void displayMessage()
23    {
24       // calls getCourseName to get the name of
25       // the course this GradeBook represents
26       System.out.printf( "Welcome to the grade book for\n%s!\n",
27          getCourseName() );
28    } // end method displayMessage
29 } // end class GradeBook
```

Fig. 3.7 | `GradeBook` class that contains a `courseName` instance variable and methods to set and get its value.

- Every instance (i.e., object) of a class contains one copy of each instance variable.
- Instance variables typically declared private.
  - Private is an access modifier.
  - Private variables and methods are accessible only to methods of the class in which they are declared.
- Declaring instance private is known as data hiding or information hiding.
- Private variables are encapsulated (hidden) in the object and can be accessed only by methods of the object's class.
  - Prevents instance variables from being modified accidentally by a class in another part of the program.
  - Set and get methods used to access instance variables.

- When a method that specifies a return type other than void completes its task, the method returns a result to its calling method.
- Method setCourseName and getCourseName each use variable courseName even though it was not declared in any of the methods.
  - Can use an instance variable of the class in each of the classes methods.
- The order in which methods are declared in a class does not determine when they are called at execution time.
- One method of a class can call another method of the same class by using just the method name.

```
1  // Fig. 3.8: GradeBookTest.java
2  // Creating and manipulating a GradeBook object.
3  import java.util.Scanner; // program uses Scanner
4
5  public class GradeBookTest
6  {
7     // main method begins program execution
8     public static void main( String[] args )
9     {
10       // create Scanner to obtain input from command window
11       Scanner input = new Scanner( System.in );
12
13       // create a GradeBook object and assign it to myGradeBook
14       GradeBook myGradeBook = new GradeBook();
15
```

```
16         // display initial value of courseName
17         System.out.printf( "Initial course name is: %s\n\n",
18            myGradeBook.getCourseName() );
19
20         // prompt for and read course name
21         System.out.println( "Please enter the course name:" );
22         String theName = input.nextLine(); // read a line of text
23         myGradeBook.setCourseName( theName ); // set the course name
24         System.out.println(); // outputs a blank line
25
26         // display welcome message after specifying course name
27         myGradeBook.displayMessage();
28      } // end main
29   } // end class GradeBookTest
```

```
Initial course name is: null

Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

Fig. 3.8 | Creating and manipulating a GradeBook object. (Part 2 of 2.)

➢ set and get methods

- A class's private fields can be manipulated only by the class's methods.
- A client of an object calls the class's public methods to manipulate the private fields of an object of the class.
- Classes often provide public methods to allow clients to set(i.e., assign values to) or get(i.e., obtain the values of) private instance variables.
- The names of these methods need not begin with set or get, but this naming convention is recommended.

Primitive Types vs. Reference Types

- Types are divided into primitive types and reference types.
- The primitive types are boolean, byte, char, short, int, long, float and double.
- All non primitive types are reference types.
- A primitive-type variable can store exactly one value of its declared type at a time.
- Primitive-type instance variables are initialized by default—variables of types byte, char, short, int, long, float and double are initialized to 0, and variables of type boolean are initialized to false.
- You can specify your own initial value for a primitive-type variable by assigning the variable a value in its declaration.

- Programs use variables of reference types (normally called references) to store the locations of objects in the computer's memory.
  - Such a variable is said to refer to an object in the program.
- Objects that are referenced may each contain many instance variables.
- Reference-type instance variables are initialized by default to the value null
  - A reserved word that represents a "reference to nothing."
- When using an object of another class, a reference to the object is required to invoke(i.e., call) its methods.
  - Also known as sending messages to an object.

Initializing Objects with Constructors

- When an object of a class is created, its instance variables are initialized by default.
- Each class can provide a constructor that initializes an object of a class when the object is created.
- Java requires a constructor call for every object that is created.
- Keyword new requests memory from the system to store an object, then calls the corresponding class's constructor to initialize the object.
- A constructor must have the same name as the class.