INTERNATIONAL UNIVERSITY

Lecture 3

Introducing the Java Class Types. Defining classes, objects and methods.

Kulbayeva Aliya Kayratovna, McS of Science
a.kulbayeva@iitu.edu.kz

Objectives

- Develop code that declares classes, interfaces, and enums, and includes the appropriate use of package and import statements.
- Develop code that declares both static and nonstatic methods. Develop code that declares and uses a variable-length argument list.
- Develop constructors for one or more of the classes. Given a class declaration, determine if a default constructor will be created, and if so, determine the behavior of that constructor. Given a nested or non-nested class listing, write code to instantiate the class.

Classes, Methods, and Interfaces

- Using Methods
- Working with Classes and Objects
- Understanding Enums
- Writing and Invoking Constructors

Using Methods

- Methods represent operations on data and also hold the logic to determine those operations
- Using methods offer two main advantages:
  - A method may be executed (called) repeatedly from different points in the program: decrease the program size, the effort to maintain the code and the probability for an error
  - Methods help make the program logically segmented, or modularized: less error prone, and easier to maintain

Defining a Method

- A method is a self-contained block of code that performs specific operations on the data by using some logic
- Method declaration:
  - Name
  - Parameter(s)
  - Argument(s)
  - Return type
  - Access modifier

Declaring Methods

- **Syntax**
- **<modifier>* <return_type> <method_name>(<argument>*){ <statement>* }**
- **Examples**

```
public class Counter{
public static final int MAX = 100;
private int value;
 public void inc(){
if( value < MAX ){
++value;
 }
} public int getValue(){
return value; } }
```

The Static Methods and Variables

- The static methods and variables are shared by all the instances of a class
- The static modifier may be applied to a variable, a method, and a block of code inside a method
- Because a static element of a class is visible to all the instances of the class, if one instance makes a change to it, all the instances see that change.

Listing 3-1. *RunStaticExample.java*

```
1. class StaticExample {
2.    static int staticCounter=0;
3.    int  counter=0;
4.    StaticExample() {
5.          staticCounter++;
6.          counter++;
7.    }
8. }
9. class RunStaticExample {
10.    public static void main(String[] args) {
11.        StaticExample se1 = new StaticExample();
12.        StaticExample se2 = new StaticExample();
13.        System.out.println("Value of staticCounter for se1: " +
           se1.staticCounter);
14.           System.out.println("Value of staticCounter for se2: " +
                   se2.staticCounter);
15.          System.out.println("Value of counter for se1: " + se1.counter);
16.          System.out.println("Value of counter for se2: " + se2.counter);
17.          StaticExample.staticCounter = 100;
18.           System.out.println("Value of staticCounter for se1: " +
                    se1.staticCounter);
19.           System.out.println("Value of staticCounter for se2: " +
                     se2.staticCounter);
20.    }
21. }
```

The Static Methods and Variables

- A static variable is initialized when a class is loaded, whereas an instance variable is initialized when an instance of the class is created
- A static method also belongs to the class. It can be called even before a single instance of the class exists
- A static method can only access the static members of the class

The Static Code Block

- A class can also have a static code block outside of any method
- The code block does not belong to any method, but only to the class executed before the class is instantiated, or even before the method main() is called

Listing 3-2. *RunStaticCodeExample.java*

```
1.  class StaticCodeExample {
2.  static int counter=0;
3.  static {
4.       counter++;
5.      System.out.println("Static Code block: counter: " + counter);
6.  }
7.  StaticCodeExample() {
8.          System.out.println("Construtor:  counter: " + counter);
9.  }
10.}
11.  public class  RunStaticCodeExample {
12.  public static void main(String[] args) {
13.        StaticCodeExample sce = new StaticCodeExample();
14.        System.out.println("main: counter:" + sce.counter);
15.  }
16.}
```

Methods with a Variable Number of Parameters

- The rules to define variable-length parameters:
  - There must be only one variable-length parameters list.
  - If there are individual parameters in addition to the list, the variable-length parameters list must appear last inside the parentheses of the method.
  - The variable-length parameters list consists of a type followed by three dots and the name.

Listing 3-3. *VarargTest.java*

```
1. import java.io.*;
2. class MyClass {
3.     public void printStuff(String greet, int... values) {
4.        for (int v : values ) {
5.           System.out.println( greet + ":" + v);
6.        }
7.     }
8.  }
9.   class VarargTest {
10.       public static void main(String[] args) {
11.         MyClass mc = new MyClass();
12.         mc.printStuff("Hello", 1);
13.         mc.printStuff("Hey", 1,2);
14.         mc.printStuff("Hey you", 1,2,3);
15.       }
16.    }
```

JavaBeans Naming Standard for Methods

- A JavaBean is a special kind of Java class that is defined by following certain rules:
  - The private variables / properties can only be accessed through getter and setter methods
  - The getter and setter methods must be public so that anyone who uses the bean can invoke them.
  - A setter method must have the void return type and must have a parameter that represents the type of the corresponding property
  - A getter method does not have any parameter and its return type matches the argument type of the corresponding setter method

JavaBeans Naming Standard for Methods

```
public class ScoreBean {
 private double meanScore;
 // getter method for property meanScore
  public double getMeanScore() {
     return meanScore;
  }
// setter method to set the value of the property meanScore
  public void setMeanScore(double score) {
     meanScore = score;
  }
}
```

- getMeanScore() and setMeanScore(), correspond to the variable (property) meanScore

Working with Classes and Objects

- A class is a template that contains the data variables and the methods that operate on those data variables following some logic
- Class members:
  - Variables represent the state of an object
  - Methods constitute its behavior

Object oriented programming

- Object Oriented Programming (OOP) a method of programming that involves the creation of intellectuals objects that model a business problem we are trying to solve .
- For example, a bank account is not something that can be physically touched but we consider the bank account as an object.
- In creating OOP we define the properties of as class of objects (all bank account) and then create individual objects from this class (your bank account)

Class

- Is a **user-defined** type

– Describes the data (attributes)
– Defines the behavior (methods)

} members

- Instances of a class are objects

Defining Classes

- The general syntax: <modifier> class <className> { }

- <className> specifies the name of the class
- class is the keyword
- <modifier> specifies some characteristics of the class:
  - Access modifiers: private, protected, default and public
  - Other modifiers: abstract, final, and strictfp

Defining Classes - Example

Listing 3-4. *ClassRoom.java*

```
1. class ClassRoom {
2 . private String roomNumber;
3.  private int totalSeats = 60;
4.  private static int totalRooms = 0;

5.   void setRoomNumber(String rn) {
6.     roomNumber = rn;
7.   }
8.   String getRoomNumber() {
9.     return roomNumber;
10.  }
11.  void setTotalSeats(int seats) {
12.      totalSeats = seats;
13.  }
14.  int getTotalSeats() {
15.     return totalSeats;
16.  }
17. }
```

Writing and Invoking Constructors

- The constructor of a class has the same name as the class and has no explicit return type
- The new operator allocates memory for the instance, and executes the constructor to initialize the memory

```
ClassRoom csLab = new ClassRoom();
```

1. Allocates memory for an instance of class ClassRoom
2. Initializes the instance variables of class ClassRoom
3. Executes the constructor ComputerLab()

Caution

- If you do not provide any constructor for a class you write, the compiler provides the default constructor for that class
- You can also define non default constructors with parameters
- The constructor may be called:
- from inside the class: from within another constructor, using this() or super()
- from outside the class: with the new operator

Key Points

- A constructor of a class has the same name as the class, and has no explicit return type
- A class may have more than one constructor
- If the programmer defines no constructor in a class, the compiler will add the default constructor with no arguments
- If there are one or more constructors defined in the class, the compiler will not provide any constructor

Key Points (cont.)

- A constructor may have zero or more parameters
- From outside the class, a constructor is always called with the new operator

Creating Objects

```
<className>  <variableName> = new <classConstructor>
```

- <variableName>: the name of the object reference that will refer to the object that you want to create
- <className>: the name of an existing class
- <classConstructor>: a constructor of the class
- The right side of the equation creates the object of the class specified by <className> with the new operator, and assigns it to <variableName> (i.e. <variableName> points to it)

Creating Objects - Example

```
class ClassRoomManager {
   public static void main(String[] args)
   {
      ClassRoom  roomOne  =  new ClassRoom();
      roomOne.setRoomNumber("MH227");
      roomOne.setTotalSeats(30);

      System.out.println("Room number: " + roomOne.getRoomNumber());
      System.out.println("Total seats: " + roomOne.getTotalSeats());
   }
}
```

Nested Class

- allows you to define a class (like a variable or a method) inside a top-level class (outer class or enclosing class)

```
class <OuterClassName> {
  // variables and methods for the outer class
  …
     class  <NestedClassName> {
        // variables and methods for the nested class
        …
      }
}
```

Nested Class

- an instance of an inner class can only exist within an instance of its outer class

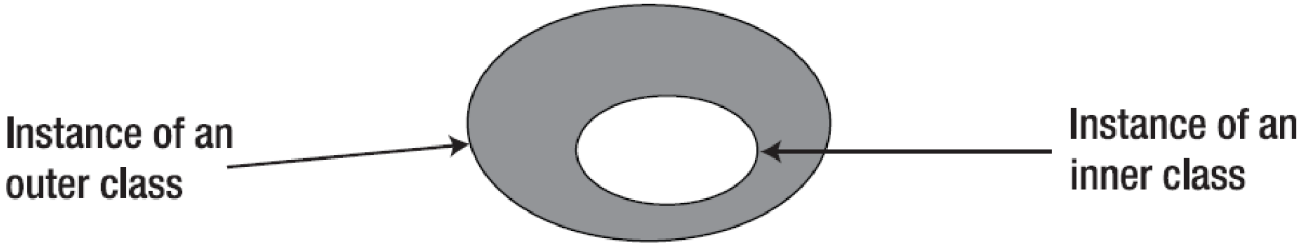

Figure 3-1. *The instance of an inner class has direct access to the instance variables and methods of an instance of the outer class.*

Listing 3-5. *TestNested.java*

```
1. class TestNested {
2.  public static void main(String[] args) {
3.    String ext = "From external class";
4.    MyTopLevel mt = new MyTopLevel();
5.    mt.createNested();
6.    MyTopLevel.MyInner inner = mt.new MyInner();
7.    inner.accessInner(ext);
8.  }
9. }
10.  class MyTopLevel{
11.    private String top = "From Top level class";
12.     MyInner minn = new MyInner();
13.     public void createNested() {
14.          minn.accessInner(top);
15.     }
16.     class MyInner {
17.         public void accessInner(String st) {
18.            System.out.println(st);
19.         }
20.      }
21.  }
```

Understanding Enums

- useful when you want a variable to hold only a predetermined set of values
- define an enum variable in two steps:
  1. Define the enum type with a set of named values
  2. Define a variable to hold one of those values

```
enum AllowedCreditCard {VISA, MASTER_CARD, AMERICAN_EXPRESS};

AllowedCreditCard visa = AllowedCreditCard.VISA;

System.out.println("The allowed credit card value: " + visa)
```

Methods of the Enum Class

- subclass of the Java class Enum

Table 3-1. *Some Methods of the Enum Class*

Method
`final boolean equals(Object obj)`
`final String name()`
`String toString()`
`static Enum valueOf(Class enumClass, String name)`

Listing 3-8. *EnumColorTest.java*

```
1. public class EnumColorTest {
2.   public static void main(String[] args) {
3.       Colors c = new Colors();
4.       c.color = Colors.ThreeColors.RED;
5.       System.out.println(c.color);
6.    }
7. }
8. class Colors {
9.     enum ThreeColors {BLUE, RED, GREEN}
10.    ThreeColors  color;
11.}
```

The output of this code follows:

```
RED
```

Thanks for attention!