Programming Fundamentals I
Java Fundamentals

Nguyen Ngoc Minh

Saigon Institute of Technology
Readings

• Readings for these lecture notes
  • Textbook:
    • Tony Gaddis & Godfrey, Starting Out with Java From Control Structures through Data Structures
  • Reference:
    • Cay S. Horstmann and Gary Cornell, Core Java™
• These lecture notes contain material © Tony Gaddis, and Godfrey Muganda, 2007
Chapter Objectives

Upon completion of this chapter, you should be able to:

• Create classes and Objects
• Understand basic concepts of Object-Oriented Programming (OOP)
Outline

• Classes and Objects
• UML
• Instance Fields and Methods
• Constructors
• Java's initialization mechanisms
• Overloading Methods and Constructors
• Scope of Instance Fields
• Packages and Import Statements
• Object-Oriented Programming
Classes and Objects

• There’re two categories of data types: class types and primitive types.
• In Java, you can develop your own class types.
• For a class type variable, we not only store data but also make request to that variable.

Ex: class String:
  • subString()
  • toUpperCase()
  • charAt()
Classes and Objects

• A class is a “blueprint” (pattern) that objects may be created from.

• A class can specify the fields and methods that a particular type of object may have.

• A class is not an object, but it can be a description of an object.

• An object created from a class using new keyword is called an instance of the class.

Ex: String name = new String(“ABC”)
Classes and Objects

Class Name

Data (Fields)

Methods
That Operate on the Data
Classes and Objects
(UML notation)

<table>
<thead>
<tr>
<th>Rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>-length</td>
</tr>
<tr>
<td>-width</td>
</tr>
<tr>
<td>+setLength()</td>
</tr>
<tr>
<td>+setWidth()</td>
</tr>
<tr>
<td>+getLength()</td>
</tr>
<tr>
<td>+getWidth()</td>
</tr>
<tr>
<td>+getArea()</td>
</tr>
</tbody>
</table>
Classes and Objects

- A **reference variable** is a variable that has data type is a class type. A reference variable contains the address of an object.
- String cityName = “Charleston”;

The object that contains the character string “Charleston”

cityName  Address to the object  Charleston
Classes and Objects

- Class objects normally have methods that perform useful operations on their data.

- Primitive variables can only store data and have no methods.
Classes and Objects

• Many objects can be created from a class.
• Each object is independent of the others.

String person = “Jenny”;
String pet = “Fido”;
String favoriteColor = “Blue”;
Classes and Objects

Classes and Objects

Classes and Objects

Person

Address

“Jenny”

Pet

Address

“Fido”

FavoriteColor

Address

“Blue”
Problem: We need to manage information of dogs. The information may include: name, and age.

- What data type is suitable to hold such information?
Classes and Objects
(Example)

```
Dog
- name
- age
+ getName()
+ getAge()
+ setName()
+ setWeight()
```
Classes and Objects
(Example)

```java
public class Dog {
    String name;
    int age;

    public String getName() {
        return name;
    }

    public void setName(String newName) {
        name = newName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```
Classes and Objects
(Example)

```java
public class temp {

    public static void main(String[] args) {
        Dog dog1, dog2;

        dog1 = new Dog();
        dog1.setName("Lu Lu");
        dog1.setAge(12);

        dog2 = new Dog();
        dog2.setName("Vang");
        dog1.setAge(5);

        System.out.println("Dog1's name: " + dog1.getName());
        System.out.println("Dog2's name: " + dog2.getName());
    }
}
```
Classes and Objects

dog1 = new Dog();

The `dog1` variable holds the address of the Dog object.
The `dog1` variable holds the address of the Dog object.

This is the state of the box object after the `setName()` method executes.
Classes and Objects

dog1.setAge(12);

The `dog1` variable holds the address of the Dog object.

This is the state of the `box` object after the `setAge()` method executes.
The *Dog* class defines the fields and methods that will exist in all objects that are instances of the *Insect* class.

The *dog1* object is an instance of the *Insect* class.

The *dog2* object is an instance of the *Insect* class.
Classes and Objects
Building a Rectangle class

• A Rectangle object will have the following fields:
  • Length: hold the rectangle’s length.
  • Width: hold the rectangle’s width.
• The Rectangle class will also have the following methods:
  • `setLength`: store a value in an object’s length field.
  • `setWidth`: store a value in an object’s width field.
  • `getLength`: return the value in an object’s length field.
  • `getWidth`: return the value in an object’s width field.
  • `getArea`: calculate & return the area of the rectangle
Classes and Objects

Rectangle class

```java
public class Rectangle {
    private float width;
    private float length;

    public float getWidth() {
        return width;
    }

    public void setWidth(float widthSize) {
        width = widthSize;
    }

    public float getLength() {
        return length;
    }

    public void setLength(float lengthSize) {
        length = lengthSize;
    }
}
```
**Classes and Objects**

**Regtangle class (Demo)**

```java
20 public class temp {
21    
22 public static void main(String[] args) {
23        Regtangle room = new Regtangle();
24
25        room.setLength(10);
26        room.setWidth(5);
27
28        System.out.println("Room's length is: " + room.getLength());
29        System.out.println("Room's width is: " + room.getWidth());
30    }
31 }
```
Classes and Objects

Header for the setLength Method

<table>
<thead>
<tr>
<th>Access specifier</th>
<th>Return Type</th>
<th>Method Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void setLength (float len)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice the word `static` does not appear in the method header designed to work on an instance of a class (`instance method`).

Parameter variable declaration
Classes and Objects

Writing and Demonstrating the getArea Method

```java
public class Rectangle {
    private float width;
    private float length;

    public float getWidth() {
        return width;
    }

    public void setWidth(float widthSize) {
    }

    public float getLength() {
        return length;
    }

    public void setLength(float lengthSize) {
    }

    public float getArea() {
        return width * length;
    }
}
```
Classes and Objects

Writing and Demonstrating the getArea Method

```java
public class temp {
    public static void main(String[] args) {
        Rectangle room = new Rectangle();
        room.setLength(10);
        room.setWidth(5);
        System.out.println("Room's area is: " + room.getArea());
    }
}
```
Classes and Objects
Access specifier

• An access specifier is a Java keyword that indicates how a field or method can be accessed.

• Public:
  • The public members can be accessed by code inside or outside the class.

• Private:
  • The private member cannot be accessed by code outside the class. The member can be accessed only by methods that are members of the same class.
Classes and Objects
Accessor and Mutator Methods

- The methods that retrieve the data of fields are called *accessors*.
- The methods that modify the data of fields are called *mutators*.
- Each field that the programmer wishes to be viewed by other classes needs an accessor.
- Each field that the programmer wishes to be modified by other classes needs a mutator.
- Other names for these methods are *getters* and *setters*. 
Classes and Objects
Accessor and Mutator Methods

• For the rectangle example, the accessors and mutators are:
  • `setLength` : Sets the value of the length field.
    ```java
    public void setLength(double len) …
    ```
  • `setWidth` : Sets the value of the width field.
    ```java
    public void setLength(double w) …
    ```
  • `getLength` : Returns the value of the length field.
    ```java
    public double getLength() …
    ```
  • `getWidth` : Returns the value of the width field.
    ```java
    public double getWidth() …
    ```
Classes and Objects
Stale Data

• If data is result of a calculation of various factors, it easily becomes *stale* should not be a data field.

• To avoid stale data, it is best to calculate the value of that data within a method rather than store it in a variable as a field.
Outline

• Classes and Objects
• **UML**
• Instance Fields and Methods
• Constructors
• Java's initialization mechanisms
• Overloading Methods and Constructors
• Scope of Instance Fields
• Packages and Import Statements
• Object-Oriented Programming
Unified Modeling Language (UML) provides a set of standard diagrams for graphically depicting object-oriented systems.

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Fields</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regtangle</td>
<td>-length</td>
<td>+setLength()</td>
</tr>
<tr>
<td></td>
<td>-width</td>
<td>+setWidth()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+getLength()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+getWidth()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+getArea()</td>
</tr>
</tbody>
</table>
UML diagrams are language independent.

UML diagrams use an independent notation to show return types, access modifiers, etc.

Access modifiers are denoted as:
- `+` public
- `-` private

Rectangle

- `width : double`

- `setWidth(w : double) : void`
UML Diagram

- UML diagrams are language independent.
- UML diagrams use an independent notation to show return types, access modifiers, etc.

Variable types are placed after the variable name, separated by a colon.

<table>
<thead>
<tr>
<th>Rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>- width : double</td>
</tr>
<tr>
<td>+ setWidth(w : double) : void</td>
</tr>
</tbody>
</table>
UML Diagram

- UML diagrams are language independent.
- UML diagrams use an independent notation to show return types, access modifiers, etc.

<table>
<thead>
<tr>
<th>Rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>- width : double</td>
</tr>
<tr>
<td>+ setWidth(w : double) : void</td>
</tr>
</tbody>
</table>

Method return types are placed after the method declaration name, separated by a colon.
• UML diagrams are language independent.
• UML diagrams use an independent notation to show return types, access modifiers, etc.

Method parameters are shown inside the parentheses using the same notation as variables.

```
Rectangle

- width : double

+ setWidth(w : double) : void
```
UML

Converting the UML Diagram to Code

- Java class file can be built easily using the UML diagram.
- The UML diagram parts match the Java class file structure.

```java
class header
{
    Fields
    Methods
}
```

```
ClassName
Fields
Methods
```
public class Rectangle
{
    private double width;
    private double length;

    public void setWidth(double w)
    {
        width = w;
    }
    public void setLength(double len)
    {
        length = len;
    }
    public double getWidth()
    {
        return width;
    }
    public double getLength()
    {
        return length;
    }
    public double getArea()
    {
        return length * width;
    }
}
Class Layout Conventions

• The layout of a source code file can vary by employer or instructor.

• Typically the layout is generally:
  • Fields are typically listed first
  • Methods are typically listed second
    • The main method is sometimes first, sometimes last.
    • Accessors and mutators are typically grouped.
Outline

- Classes and Objects
  - UML
  - Instance Fields and Methods
- Constructors
- Java's initialization mechanisms
- Overloading Methods and Constructors
- Scope of Instance Fields
- Packages and Import Statements
- Object-Oriented Programming
Instance Fields and Methods

- Fields and methods that are declared as previously shown are called *instance fields* and *instance methods*.
- Objects created from a class each have their own copy of instance fields.
- Instance Field/Methods are **NOT** declared with a special keyword, *static*.
- Instance fields and instance methods require an object to be created in order to be used.
Instance Fields and Methods
Initialization of Instance Fields

• Local variables must be initialized values before being used.

• Instance fields are automatically initialized to default values by Java's initialization mechanism.

• Initialization of Instance Fields happens before
Instance Fields and Methods

Example

```java
import javax.swing.JOptionPane;
public class RoomAreas {

    public static void main(String[] args) {
        Rectangle kitchen, bedroom;
        kitchen = new Rectangle();
        kitchen.setLength(10);
        kitchen.setWidth(5);

        bedroom = new Rectangle();
        bedroom.setLength(7);
        bedroom.setWidth(4);

        JOptionPane.showMessageDialog(null, "The kitchen's area is " +
                           kitchen.getArea() + "\n" +
                           "The bedroom's area is " + bedroom.getArea());

        System.exit(0);
    }
}
```
Instance Fields and Methods
States of Two Different Rectangle Objects

The kitchen variable holds the address of a Rectangle Object.

The bedroom variable holds the address of a Rectangle Object.
Outline

- Classes and Objects
  - UML
  - Instance Fields and Methods
  - **Constructors**
  - Java's initialization mechanisms
  - Overloading Methods and Constructors
  - Scope of Instance Fields
  - Packages and Import Statements
  - Object-Oriented Programming
Constructors

- Constructors have a few special properties that set them apart from normal methods.
  - Constructors have the same name as the class.
  - Constructors have no return type (not even void).
  - Constructors may not return any values.
  - Constructors are typically public.
Constructors
Default constructor

• When an object is created, its constructor is always called.

• If you do not write a constructor, Java provides one when the class is compiled. The constructor that Java provides is known as the default constructor.

• Constructor is used to initialize data fields of new objects.

• Constructor is one of Java’s mechanisms used to manage initialization.
Constructors

Default constructor

- The default constructor is a constructor with no parameters, used to initialize an object in a default configuration.
- The **only** time that Java provides a default constructor is when you do not write **any** constructor for a class.
- A default constructor is **not** provided by Java if a constructor is already written.
Constructors
Writing Your Own No-Arg Constructor

• A constructor that does not accept arguments is known as a no-arg constructor.
• The default constructor (provided by Java) is a no-arg constructor.
• We can write our own no-arg constructor
  ```java
  public Rectangle()
  {
    length = 1.0;
    width = 1.0;
  }
  ```
Constructors
Constructor with parameters

• One of the String class constructors accepts a String literal as an argument.
• This String literal is used to initialize a String object.
• For instance:

  String name = new String(“Michael Long”);
Constructors
Constructor with parameters

/**
   Constructor
   @param len The length of the rectangle.
   @param w The width of the rectangle.
*/
public Rectangle(double len, double w) {
    length = len;
    width = w;
}
Constructors

Constructors in UML

- In UML, the most common way constructors are defined is:

<table>
<thead>
<tr>
<th>Rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>- width : double</td>
</tr>
<tr>
<td>- length : double</td>
</tr>
</tbody>
</table>

+Rectangle(len:double, w:double):

+ setWidth(w : double) : void
+ setLength(lcn : double): void
+ getWidth() : double
+ getLength() : double
+ getArea() : double

Notice there is no return type listed for constructors.
Outline

- Classes and Objects
  - UML
  - Instance Fields and Methods
  - Constructors
  - Java's initialization mechanisms
- Overloading Methods and Constructors
- Scope of Instance Fields
- Packages and Import Statements
- Object-Oriented Programming
Java's initialization mechanisms

- Local variables must be initialized before being used.
- Three initialization mechanism: *instance initializers*, *instance variable initializers*, and *constructors*.
- All three mechanisms perform automatically when an object is created.
- When an object is created by new keyword, the instance variables will be assigned default values before any constructor is called.
  - It sets all of the object’s numeric fields to 0.
  - It sets all of the object’s boolean fields to false.
  - It sets all of the object’s reference variables to the special value *null*. 
Java's initialization mechanisms
Instance variable initializers

```java
public class Rectangle {
    public float width = 2;
    public float length = 3;

    public Rectangle() {
    }
}
```
Java's initialization mechanisms

Instance initializers

```java
public class Rectangle {

    public float width;
    public float length;

    {
        width = 2;
        length = 3;
    }

    public Rectangle(){
    }
}
```
Java's initialization mechanisms
Order of Initialization

- Instance initializers, and instance variable initializers happen first, constructor is executed later.

```java
class Rectangle {
    public float width = 1;
    public float length = 1;

    { 
        width = 2;
        length = 2;
    }

    public Rectangle()
    { 
        width = 3;
        length = 3;
    }
}
```
Outline

Classes and Objects
- UML
- Instance Fields and Methods
- Constructors
- Java's initialization mechanisms
- Overloading Methods and Constructors
- Scope of Instance Fields
- Packages and Import Statements
- Object-Oriented Programming
Constructors
Overloading Methods and Constructors

- Two or more methods in a class may have the same name as long as their parameter lists are different.
- When this occurs, it is called method overloading. This also applies to constructors.
- Method overloading is important because sometimes you need several different ways to perform the same operation.
If we were to add the no-args constructor we wrote previously to our Rectangle class in addition to the original constructor we wrote, what would happen when we execute the following calls?

```java
Rectangle box1 = new Rectangle();
Rectangle box2 = new Rectangle(5.0, 10.0);
```
Constructors
Method Signature and Binding

• A **method signature** consists of the method’s name and the data types of the method’s parameters, in the order that they appear. The return type is **not** part of the signature.

• The process of matching a method call with the correct method is known as **binding**. The compiler uses the method signature to determine which version of the overloaded method to bind the call to.
Constructors

Method Signature and Binding (Example)

```java
public int add(int num1, int num2) {
    int sum = num1 + num2;
    return sum;
}
public String add(String str1, String str2) {
    String combined = str1 + str2;
    return combined;
}
```
Constructors
Overloading Methods and Constructors (Example)

Overloaded Constructors

Overloaded deposit methods

Overloaded withdraw methods

Overloaded setBalance methods

BankAccount

- balance: double

+ BankAccount():
+ BankAccount(startBalance: double):
+ BankAccount(str: String):

+ deposit(amount: double): void
+ deposit(str: String): void
+ withdraw(amount: double): void
+ withdraw(str: String): void
+ setBalance(b: double): void
+ setBalance(str: String): void
+ getBalance(): double
Outline

Classes and Objects
• UML
• Instance Fields and Methods
• Constructors
• Java's initialization mechanisms
• Overloading Methods and Constructors
• **Scope of Instance Fields**
• Packages and Import Statements
• Object-Oriented Programming
Scope of Instance Fields

- Variables declared as instance fields in a class can be accessed by any instance method in the same class as the field.
- If an instance field is declared with the public access specifier, it can also be accessed by code outside the class.
Scope of Instance Fields
Shadowing

• Within a method, variable names must be unique.
• A method may have a local variable with the same name as an instance field.
• The local variable will hide the value of the instance field.
• This is called shadowing.
• Shadowing is discouraged and local variable names should not be the same as instance field names.
• To refer to instance field in case of shadowing, use key word this
Outline

- Classes and Objects
  - UML
  - Instance Fields and Methods
  - Constructors
  - Java's initialization mechanisms
  - Overloading Methods and Constructors
  - Scope of Instance Fields
- Packages and Import Statements
- Object-Oriented Programming
Packages and Import Statements

- Classes in the Java API are organized into packages
- Explicit and Wildcard import statements
  - Explicit imports name a specific class
    - import java.util.Scanner;
  - Wildcard imports name a package, followed by an *
    - import java.util.*;
- The java.lang is automatically made available to any Java class
## Some Java Standard Packages

Table 6-2

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.applet</td>
<td>Provides the classes necessary to create an applet.</td>
</tr>
<tr>
<td>java.awt</td>
<td>Provides classes for the Abstract Windowing Toolkit. These classes are used in drawing images and creating graphical user interfaces.</td>
</tr>
<tr>
<td>java.io</td>
<td>Provides classes that perform various types of input and output.</td>
</tr>
<tr>
<td>java.lang</td>
<td>Provides general classes for the Java language. This package is automatically imported.</td>
</tr>
<tr>
<td>java.net</td>
<td>Provides classes for network communications.</td>
</tr>
<tr>
<td>java.security</td>
<td>Provides classes that implement security features.</td>
</tr>
<tr>
<td>java.sql</td>
<td>Provides classes for accessing databases using structured query language.</td>
</tr>
<tr>
<td>java.text</td>
<td>Provides various classes for formatting text.</td>
</tr>
<tr>
<td>java.util</td>
<td>Provides various utility classes.</td>
</tr>
<tr>
<td>javax.swing</td>
<td>Provides classes for creating graphical user interfaces.</td>
</tr>
</tbody>
</table>
Some Important Points

• Each file .java can contain one/many classes but only one public class.
• Public class must have the same name with filename.
• A class can be defined inside another class definition. We call this class **Inner Class**
Inner Class

```java
public class Outer {
    Data Fields
    methods

    private class Outer{
        Data Fields
        methods
    }
}
```
Comparing Objects

- The `==` operator only compare the address of objects.
- We have to compare the content of object to know if two objects equal.
- Example Student.java
Outline

Classes and Objects
• UML
• Instance Fields and Methods
• Constructors
• Java's initialization mechanisms
• Overloading Methods and Constructors
• Scope of Instance Fields
• Packages and Import Statements
• **Object-Oriented Programming**
Object-Oriented Programming

- Object-oriented programming is centered on creating objects rather than procedures.
- Everything is object.
An Everyday Example of an Object—An Alarm Clock

- Fields define the state that the alarm is currently in.
  - The current second (a value in the range of 0-59)
  - The current minute (a value in the range of 0-59)
  - The current hour (a value in the range of 1-12)
  - The time the alarm is set for (a valid hour and minute)
  - Whether the alarm is on or off (“on” or “off”)
An Everyday Example of an Object—An Alarm Clock

Methods are used to change a field’s value

- Set time
- Set alarm time
- Turn alarm on
- Turn alarm off
- Increment the current second
- Increment the current minute
- Increment the current hour
- Sound alarm

Public methods are accessed by users outside the object.

Private methods are part of the object’s internal design.
Object-Oriented Programming

- Object-oriented programming combines data and behavior via *encapsulation*.
- *Data hiding* is the ability of an object to hide data from other objects in the program.
Object-Oriented Programming
Data hiding

• Only an object’s methods should be able to directly manipulate its data.
• Other objects are allowed to manipulate an object’s data via the object’s methods.
Object-Oriented Programming

Data hiding

Code Outside the Object

Object

Data (Fields)
typically private to this object

Methods That Operate on the Data