

Java Programming, Comprehensive

Lecture 3

Bineet Sharma

Agenda

- ▶ **Object Oriented Programming Concepts**
 - ▶ Thinking in objects
 - ▶ Abstraction
 - ▶ Encapsulation
 - ▶ Class and Object
 - ▶ UML diagram
 - ▶ Writing classes
- ▶ **String**
 - ▶ Exercise

Object-Oriented Programming: Concepts

- ▶ Thinking in nouns enables you to represent elements of a problem domain into a solvable software entities – objects
- ▶ Programs (applications) are collections of interacting objects
- ▶ Writing Object Oriented Programming (OOP) involves identifying those objects and programming them
- ▶ A **home** is a good real world example of OOP
- ▶ Language like Java, C++ provides rich set of OOP features
- ▶ Defining objects involves identifying
 - ▶ State (data – primitive or reference)
 - ▶ Behaviors (functionality)

Object-Oriented Programming: Concepts

► Thinking in Objects:

- Everything is an object. Any conceptual component in the problem domain (dog, geometry, services, tax, etc.) can be represented into an object
- Example: A light bulb operation is represented into a Java object

Type	Bulb
Behavior	turnOn()
	turnOff()
	turnBrightnessDown()
	turnBrightnessUp()



- An OO Program is really a bunch of objects telling each other what to do by sending messages to each other
- Each object has its own memory and contains data or other objects and interacts with other objects by sending or receiving messages

Object-Oriented Programming: Concepts

- ▶ **Example of objects (has states and behaviors)**

- ▶ **Dog**

- ▶ State: color, breed, age
 - ▶ Behavior: wagtail, fetch, sit

- ▶ **Student**

- ▶ State: name, ID, DOB, major, minor
 - ▶ Behavior: find grade, find current class, pay dues

- ▶ **Scanner**

- ▶ State: input stream (File, System.in)
 - ▶ Behavior: get string, get integer

- ▶ **String**

- ▶ State: Stream of characters
 - ▶ Behavior: get substring, change case

Object-Oriented Programming: Concepts

- ▶ Example of OOP (Object Oriented Program), which have objects interacting (passes messages) with each other:
 - ▶ Home has:
 - ▶ Wife, Dog, Kids, Kitchen, House, Car, Computer, Office Room, Husband
 - We do not worry about internals of a home, **do we?**
 - ▶ Bank has:
 - ▶ Tellers, Customers, Computers, Accounts
 - ▶ Customer Relationship Management has:
 - ▶ Salesman, Customer, Dealer, Contact Information, Campaign, Messages, Reports
 - ▶ Design and Drawing has:
 - ▶ Shapes, Brush, Pen, Canvas, Resolution, Co-ordinate system, Layout
 - ▶ Classroom has:
 - ▶ Teacher, Student, Computers, Desks, Projector, Screen
 - We do not worry about internals of workings of a class room, **do we?**
 - We are only interested in the purpose of a class room

Object-Oriented Programming: Concepts

- ▶ Thinking in objects....
 - ▶ How do you identify such objects?
 - ▶ Go through OOP Design.pdf for quick review
 - ▶ How do you create such objects? Do objects have types?
 - ▶ You can create new objects using same group of information again and again from a template – called **class**
 - ▶ A class is really a **new type** in Java. So, just like using a primitive data type (char, int, etc), you use class, to create an object
int age = 35;
Product toaster = new Product("toaster");
- ▶ Basically class concept enables you to extend Java language by defining your own type to suit your own need to solve a problem
- ▶ Java language provides same functionality to your new types as it does for primitive data types – that is HUGE!

Object-Oriented Programming: Concepts

▶ Thinking in objects....

- ▶ So, really, a class is reusable software component.
 - ▶ And an object is instance of that class with concrete data in memory – Avatar!
- ▶ Each object keeps internal information (**state**) hidden from others in **instance variables** and
 - ▶ reacts through, **behaviors**, to external messages sent, through **methods**
- ▶ Each object of a same type (class) reacts to a message in a similar fashion (this is loaded concept).
 - ▶ So, once a class is established, you can make as many objects of that class as you like, and then manipulate those objects as they are the elements of the problem you are trying to solve.

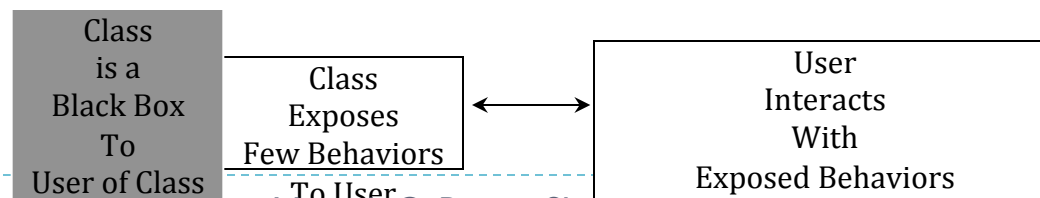
Object-Oriented Programming: Concepts

► Distinction between Class and Objects

Class	Object
Type	Variable
One	Many
Abstract	Real
Ex: Product	Ex: emersonMicrowave
Dog	blackLab
Teacher	javaInstructor

Object-Oriented Programming: Concepts

- ▶ **Abstraction** (used as noun, and verb both):
 - ▶ **As entity**, is the representation of an element of the problem domain in the computer program. Light bulb ex.
 - ▶ **As technique**, it helps in identifying which info to hide
- ▶ **Information Hiding: Separation of class implementation from the use of the class**
 - ▶ Developer of the class provides a description of the class and let the user know how the class can be used
 - ▶ The user of the class does not need to know the details
- ▶ **Encapsulation: The outcome of enclosing state and internal behaviors of the class hidden (bundling & hiding)**



Object-Oriented Programming: Concepts

- ▶ Abstraction, information hiding and encapsulation can get confusing
 - ▶ They are different but very related concepts, summarizing
 - ▶ **Abstraction** is a technique which helps identify which information can be hidden
 - ▶ **Encapsulation** is the result of information hiding
- ▶ *Encapsulation is the result of hiding a representation and implementation in an object. The representation is not visible and cannot be accessed directly from outside the object. Operation is the only way to access and modify an object's operation – Design Patterns - GOF*

Object-Oriented Programming: Concepts

- ▶ How do you design a class?
- ▶ Class should describe a single entity. Ex. Student class
 - ▶ Student class should not have any functionality of a Staff!
- ▶ At times you find too many responsibilities to incorporate in a class
 - ▶ Time to break them apart into multiple classes sharing responsibilities
 - ▶ Example: String classes
 - ▶ String: deals with immutable strings
 - ▶ StringBuilder: deals with mutable strings
 - ▶ StringBuffer: like StringBuilder which has synchronized methods for the updates

Object-Oriented Programming: Concepts

- ▶ **Classes are designed for reuse purposes. Users can use your classes in many different combinations, orders, situation and environments, So, your class designs should:**
 - ▶ Not impose any restrictions on what they can do with it
 - ▶ Have properties which can be set in any order
 - ▶ Have any combination of values
 - ▶ Design methods which operates independently of their order of occurrence
- ▶ **Your home, startup company functions as a OOP**
 - ▶ However, a home/startup company can start crumbling as well, if objects start behaving erratically (NOOP)
- ▶ **How to fix it?**
 - ▶ Improvise and redesign

Object-Oriented Programming: Concepts

▶ Classes include specifications of:

▶ Data

- ▶ Instance variables - type and visibility - to include in objects.
 - Object data has no static modifier

▶ Behaviors

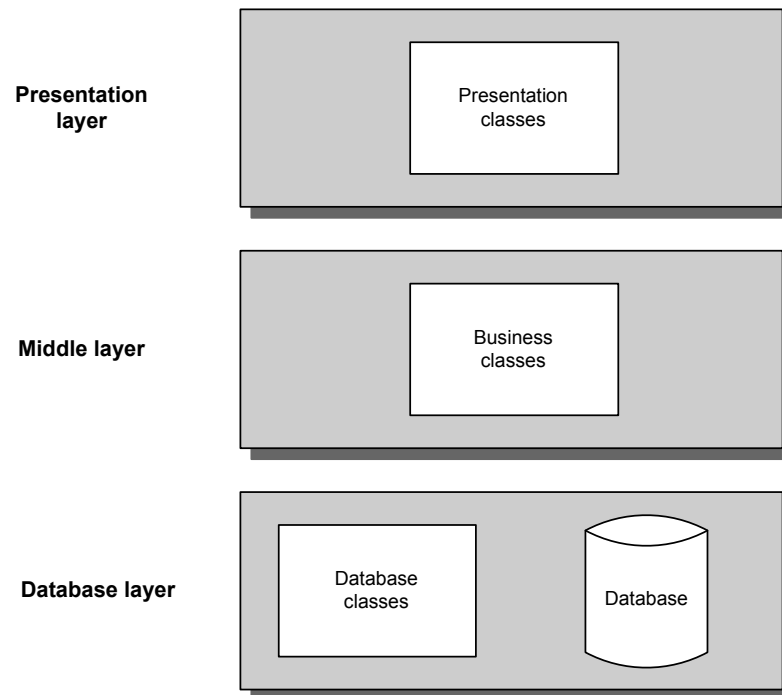
- ▶ Implementations of methods – type and visibility – to include in objects
 - Object data has no static modifier

▶ Classes also includes:

- ▶ Static variables
 - class state
- ▶ Static methods
 - class behavior
- ▶ Inner classes
- ▶ Enums

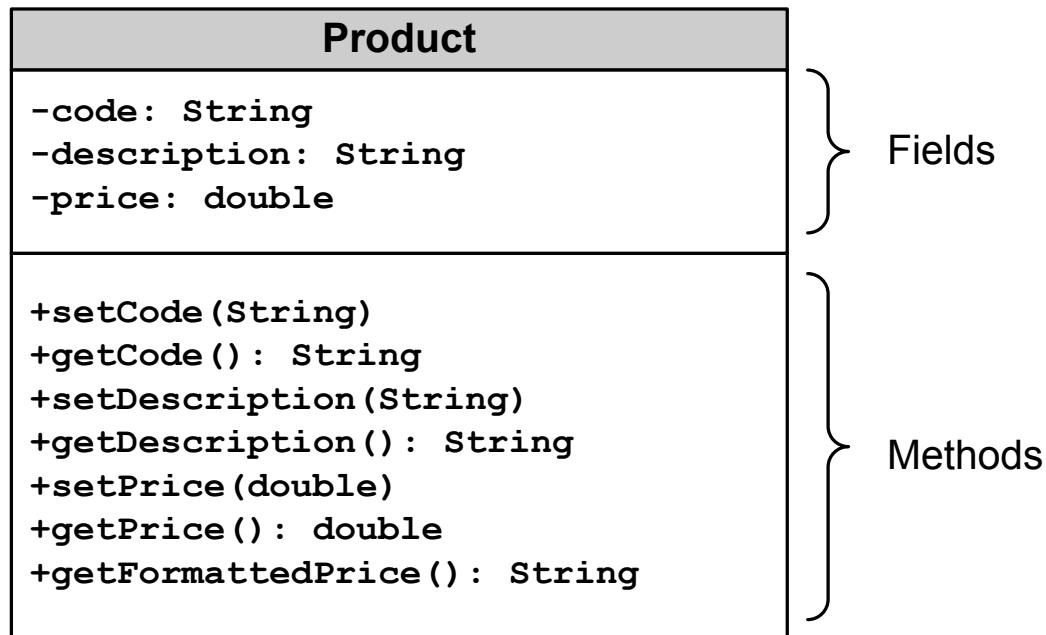
Object-Oriented Programming: Concepts

- ▶ A real world application is modular and is spread out in multilayers, each layer has multiple classes
- ▶ Example of an architecture of a three-tiered application



Object-Oriented Programming: Class

- ▶ Designing a Product application
 - ▶ Product has product code, description, price.
 - ▶ Customer interacts with Product
- ▶ What would a Product class look like? Use UML diagram to describe your concept



Object-Oriented Programming: Class

► Writing UML diagram

- *UML (Unified Modeling Language)* is the industry standard used to describe the classes and objects of an object-oriented application.
- The minus sign (-) in a UML *class diagram* marks the fields and methods that can't be accessed by other classes, while the plus sign (+) marks the fields and methods that can be accessed by other classes.
- For each field, the name is given, followed by a colon, followed by the data type.
- For each method, the name is given, followed by a set of parentheses. If a method requires parameters, the data type of each parameter is listed in the parentheses. Otherwise, the parentheses are left empty, and the data type of the value that's going to be returned is given after the colon.

Object-Oriented Programming: Class

- ▶ When writing a class you would (normally):
 - ▶ Write multiple instance variables and methods
 - ▶ Write special method, called constructors, to initialize instance variables and avoid errors
 - ▶ Write a public no-arg constructor
 - ▶ Override the **equals**, **toString** methods defined in **Object**
 - ▶ Follow standard Java programming style and naming conventions
 - ▶ Choose informative names of classes, data fields, and methods
 - ▶ Place instance variables before constructors
 - ▶ Place constructors before the methods

Object-Oriented Programming: Class

- ▶ Let us get started
- ▶ Identify an object in the problem domain

```
Welcome to the Product Selector

Enter product code: java

SELECTED PRODUCT
Description: Murach's Beginning Java
Price:      $49.50

Continue? (y/n):
```

- ▶ What would be a data type (class) needed for this type of program output?
- ▶ More specifically, do you need a new type? A class?

Object-Oriented Programming: Class

- ▶ Once you identify the class and its state and behavior, use your IDE to create the class or write manually
- ▶ Example of a class. Your type, the *product* class:

```
import java.text.NumberFormat;

public class Product
{
    // the instance variables
    private String code;
    private String description;
    private double price;

    // the constructor
    public Product()
    {
        code = "";
        description = "";
        price = 0;
    }
}
```

Object-Oriented Programming: Class

► The product class (cont.):

```
// the set and get methods for the code variable
public void setCode(String code)
{
    this.code = code;
}
public String getCode()
{
    return code;
}

// the set and get methods for the description variable
public void setDescription(String description)
{
    this.description = description;
}
public String getDescription()
{
    return description;
}
```

Object-Oriented Programming: Class

► The product class (cont.):

```
// the set and get methods for the price variable
public void setPrice(double price)
{
    this.price = price;
}
public double getPrice()
{
    return price;
}

// a custom get method for the price variable
public String getFormattedPrice()
{
    NumberFormat currency =
        NumberFormat.getCurrencyInstance();
    return currency.format(price);
}
}
```

Object-Oriented Programming: Class

- ▶ Add instance variables:

The syntax for declaring instance variables

```
public|private primitiveType|ClassName variableName;
```

Examples

```
private double price;  
private int quantity;  
private String code;  
private Product product;
```

Object-Oriented Programming: Class

► Add instance variables:

Where can you declare instance variables?

```
public class Product
{
    //common to code instance variables here
    private String code;
    private String description;
    private double price;
    //the constructors and methods of the class
    public Product(){}
    public void setCode(String code){}
    public String getCode(){ return code; }
    public void setDescription(String description){}
    public String getDescription(){ return description; }
    public void setPrice(double price){}
    public double getPrice(){ return price; }
    public String getFormattedPrice(){
        return formattedPrice; }

    //also possible to code instance variables here
    //not a brilliant idea though
    private int test;
}
```


Object-Oriented Programming: Class

- ▶ Constructors are special method invoked to create objects when **new** is used. Constructors:
 - ▶ have same name as class
 - ▶ do not have a return type – not even void
- ▶ Constructors with no parameter are called no-arg (default) constructors

The syntax for coding constructors

```
public ClassName([parameterList]){  
    // the statements of the constructor  
}
```

A constructor that assigns default values

```
public Product(){  
    code = "";  
    description = "";  
    price = 0.0;  
}
```

Object-Oriented Programming: Class

► Add constructors to your class :

A custom constructor with three parameters

```
public Product(String code, String description,  
                double price){  
    this.code = code;  
    this.description = description;  
    this.price = price;  
}
```

Another way to code same constructor

```
public Product(String c, String d, double p){  
    code = c;  
    description = d;  
    price = p;  
}
```

Another constructor uses **ProductDB** to get details

```
public Product(String c){  
    code = c;  
    Product p = ProductDB.getProduct(c);  
    description = p.getDescription();  
    price = p.getPrice();  
}
```

Object-Oriented Programming: Class

- ▶ Add constructors to your class:
 - ▶ A class can have many constructors
 - ▶ A class can have no constructor as well
 - ▶ In this case, a no-arg constructor with an empty body is implicitly defined in the class by Java.
 - ▶ This constructor is called a *default constructor*
 - ▶ It is provided automatically *only if no constructors are explicitly defined in the class.*
- ▶ If you defined a constructor in your class and no default constructor, then a code like this is not possible
`MyClass myObject = new MyClass();`
- ▶ What is rational behind multiple constructors?

Object-Oriented Programming: Class

- ▶ Next, define behaviors by adding as many methods as needed:

The syntax for coding a method

```
public|private returnType methodName([parameterList])
{
    // the statements of the method
}
```

A method that doesn't accept parameters or return data

```
public void printToConsole()
{
    System.out.println(
        code + "|" + description + "|" + price);
}
```

Object-Oriented Programming: Class

► Add methods:

A get method that returns a string

```
public String getCode()  
{  
    return code;  
}
```

A get method that returns a double value

```
public double getPrice()  
{  
    return price;  
}
```

A custom get method

```
public String getFormattedPrice()  
{  
    NumberFormat currency =  
        NumberFormat.getCurrencyInstance();  
    return currency.format(price);  
}
```

Object-Oriented Programming: Class

► Add methods:

A set method

```
public void setCode(String code)
{
    this.code = code;
}
```

Another way to code a set method

```
public void setCode(String productCode)
{
    code = productCode;
}
```

Object-Oriented Programming: Class

► Add methods (contd.):

A method that accepts one argument

```
public void printToConsole(String sep)
{
    System.out.println(
        code + sep + description + sep + price);
}
```

An overloaded method that provides a default value

```
public void printToConsole()
{
    // call the method that accepts an argument
    printToConsole("|");
}
```

Object-Oriented Programming: Class

► Add methods (contd.):

An overloaded method with two arguments

```
public void printToConsole(  
    String sep, boolean printLineAfter)  
{  
    // call the method that accepts one argument  
    printToConsole(sep);  
    if (printLineAfter)  
        System.out.println();  
}
```

Code that calls the printToConsole methods

```
Product p = ProductDB.getProduct("java");  
  
p.printToConsole();  
p.printToConsole(" ", true);  
p.printToConsole(" ");
```

The console output

```
java|Murach's Beginning Java 2|49.5  
java    Murach's Beginning Java  49.5  
java    Murach's Beginning Java  49.5
```


Object-Oriented Programming: Class

- ▶ What is method overloading?
- ▶ How compiler knows which method to call?
- ▶ Method has following components:
 - ▶ **Accessor** type (public/private/protected/<default>)
 - ▶ **Return** type (primitive or reference)
 - ▶ **Static** or not (can be called using class or object)
 - ▶ Optional **parameter**(s)/argument(s) (input to the method)
- ▶ Which of above participate in method overloading?
 - ▶ Method Signatures: Only two of the above components participate in method overloading
 - Method name, order and type of parameters

Object-Oriented Programming: Class

▶ The **this** keyword:

- ▶ Is the name of a reference that refers to an object itself
 - ▶ A way to refer to an object during the construction of the object
 - Meaning, while object is being constructed and there is no reference created yet for that object
 - ▶ Myself!
- ▶ Usually you use **this** keyword to:
 - ▶ refer a class's hidden data fields (private variables), within the same class
 - ▶ Enable a constructor of same class to invoke another constructor of same class

Object-Oriented Programming: Class

► The syntax of using the **this** keyword

```
// refer to an instance variable of the current object  
this.variableName
```

```
// call another constructor of the same class  
this(argumentList);
```

```
// call a method of the current object  
this.methodName(argumentList)
```

```
// pass the current object to a method  
objectName.methodName(this)
```

```
// pass the current object to a static method  
ClassName.methodName(this)
```

► Why we really need a 'this'?

Object-Oriented Programming: Class

► How to refer to instance variables in a method?

```
public Product( String code, String description,  
               double price){  
    this.code = code;  
    this.description = description;  
    this.price = price;  
}
```

► How to refer to methods

```
public String getFormattedPrice()  
{  
    NumberFormat currency =  
        NumberFormat.getCurrencyInstance();  
    return currency.format(this.getPrice());  
}
```

Object-Oriented Programming: Class

How to call a constructor

```
public Product()  
{  
    this("", "", 0.0);  
}
```

How to send the current object to a method

```
public void print()  
{  
    System.out.println(this);  
}
```

How to send the current object to a static method

```
public void save()  
{  
    ProductDB.saveProduct(this);  
}
```

Object-Oriented Programming: Class

► Using a class to create objects:

Syntax

```
ClassName variableName;  
variableName = new ClassName(argumentList);  
or  
ClassName variableName = new ClassName(argumentList);
```

Example 1: No arguments

```
Product myProduct;  
myProduct = new Product()  
or  
Product myProduct = new Product();
```

Example 2: One literal argument

```
Product myProduct = new Product("java");
```

Example 3: One variable argument

```
Product myProduct = new Product(productCode);
```

Example 4: Three arguments

```
Product myProduct =  
    new Product(code, description, price);
```

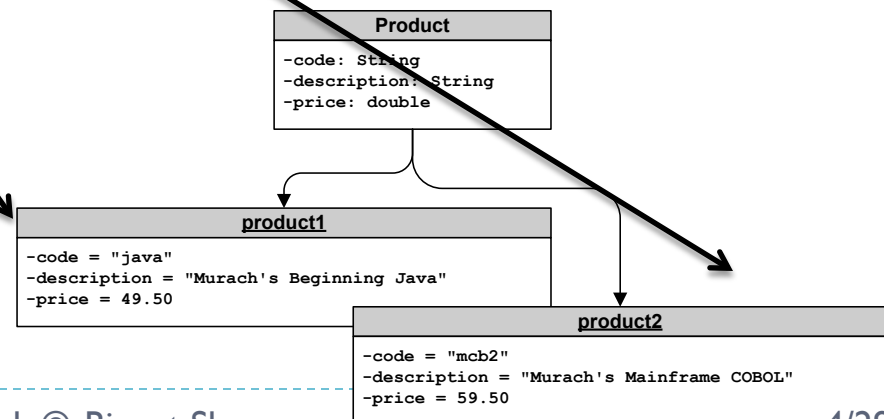
Object-Oriented Programming: Class

► What happens when you call a **new**?

Example 5: Three arguments

```
Product product1 = new Product("java", "Murach\'s  
Beginning Java", 49.50);  
  
Product product2 = new Product("mcb2", "Murach\'s  
Mainframe COBOL", 59.50);  
  
/*new creates memory for instance of Product class,  
called object, calls the specified constructor and, its  
reference is assigned to the variable  
*/
```

► The relationship between a class (Product) and its objects (product1 and product2)



Object-Oriented Programming: Class

► Calling methods on an object

Syntax

```
objectName.methodName(argumentList)
```

Example 1: Sends and returns no arguments

```
product.printToConsole();
```

Example 2: Sends no arguments and returns a double value

```
double price = product.getPrice();
```

Example 3: Sends an argument and returns a String object

```
String formattedPrice =  
    product.getFormattedPrice(includeDollarSign);
```

Example 4: A method call within an expression

```
String message =  
    "Code: " + product.getCode() + "\n\n" +  
    "Press Enter to continue or enter 'x' to exit:";
```


Object-Oriented Programming: Class

- ▶ When you pass a primitive type to a method, a copy of the data is passed to the method

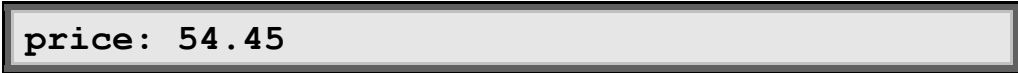
A method that changes the value of a double type

```
static double increasePrice(  
    double price) // returns a double  
{  
    return price *= 1.1;  
}
```

Code that calls the method

```
double price = 49.5;  
double newPrice = increasePrice(price); //reassignment  
System.out.println("price: " + newPrice);
```

Result



price: 54.45

- ▶ The price in calling routine stays as 49.5. price of calling routine and increasePrice() method are not same entity

Object-Oriented Programming: Class

► A **reference** type that's passed to a method

A method that changes a value stored in a Product object

```
static void increasePrice(  
    Product product) // no return value  
{  
    double price = product.getPrice();  
    product.setPrice(price *= 1.1);  
}
```

Code that calls the method

```
Product product = ProductDB.getProduct("java");  
System.out.println("product.getPrice(): " +  
    product.getPrice());  
increasePrice(product); // no reassignment necessary  
System.out.println("product.getPrice(): " +  
    product.getPrice());
```

Result

```
product.getPrice(): 49.5  
product.getPrice(): 54.45
```

► Here, the changes made in product reference by increasePrice() method is reflected in the calling routine

Object-Oriented Programming: Class

- ▶ Swapping two values in Java: Primitive type and Reference Type

Swap Methods

```
// swaps two primitive integer
private static void swapValue(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}

// swaps two String. What are we swapping here?
private static void swapValue(String one, String two)
{
    String tempStr = one;
    one = two;
    two = tempStr;
}
```

Object-Oriented Programming: Class

- ▶ Swapping two values in Java: Primitive type and Reference Type

Swap Methods

```
// swaps the two point references
private static void swapValue(Point point1, Point
                                point2)
{
    Point tempPoint = point1;
    point1 = point2;
    point2 = tempPoint;
}

// swaps the x,y co-ordinate of a point reference
private static void swapValue(Point p1)
{
    int temp = p1.x;
    p1.x = p1.y;
    p1.y = temp;
}
```

Object-Oriented Programming: Class

► Swapping two values in Java: Primitive type and Reference Type

Example of swapping

```
int x = 15, y = 20;
swapValue(x, y);
System.out.println("After swap call x: " + x
    + " y: " + y);

x = 15; y = 20; // reset the value
Point p = new Point(x, y);
swapValue(p); //overloaded method
System.out
    .println("After swap method call Point p: " +p)

String str1 = "John";
String str2 = "Elizabeth";

swapValue(str1, str2); //overloaded method
System.out
    .println("After swap method call String str1:
        " + str1 + " str2: " + str2);
```

Object-Oriented Programming: Class

► Swapping two values in Java: Primitive type and Reference Type

Example of swapping

```
int x = 15, y = 20;
swapValue(x, y);
System.out.println("After swap call x: " + x
    + " y: " + y);

x = 15; y = 20; // reset the value
Point p = new Point(x, y);
swapValue(p); //overloaded method
System.out
    .println("After swap method call Point p: " + p)

String str1 = "John";
String str2 = "Elizabeth";

swapValue(str1, str2); //overloaded method
System.out
    .println("After swap method call String str1:
        " + str1 + " str2: " + str2);
```

```
After swap method call x: 15 y: 20
After swap method call Point p:
    java.awt.Point[x=20,y=15]
After swap method call String str1: John
    str2: Elizabeth
```

Object-Oriented Programming: Class

- ▶ You can pass data to a method:
 - ▶ As pass by value, or
 - ▶ As pass by reference
- ▶ So, is Java pass by reference or pass by value, or both?
 - ▶ Java is strictly pass by value, even for reference type
- ▶ So, it is better to write:
 - ▶ Pass by value: a primitive type
 - ▶ Pass by value: a reference type
- ▶ When you pass by reference:
 - ▶ The changes in reference itself is not reflected in calling routine
 - ▶ But the changes in the data of reference is reflected in the calling routine, as the called routine directly works on that reference

Object-Oriented Programming: Class

- ▶ Code walk through – ProductApp Class that uses ProductDB class

```
Welcome to the Product Selector

Enter product code: java

SELECTED PRODUCT
Description: Murach's Beginning Java
Price:      $49.50

Continue? (y/n):
```

- The ProductDB class simulates the processing that would be done by a database class. In a more realistic application, this class would retrieve the data for a product from a file or database and then fill the Product object with that data. It would also include methods for adding, modifying, and deleting existing products.

Object-Oriented Programming: Class

▶ Working with static fields:

Declaring static fields

```
private static int numberOfObjects = 0;  
private static double majorityPercent = .51;  
public static final int DAYS_IN_JANUARY = 31;  
public static final float EARTH_MASS_IN_KG = 5.972e24F;
```

Object-Oriented Programming: Class

► Working with static fields:

A class that contains a static constant and a static method

```
public class FinancialCalculations
{
    public static final int MONTHS_IN_YEAR = 12;

    public static double calculateFutureValue(
        double monthlyPayment,
        double yearlyInterestRate,
        int years)
    {
        int months = years * MONTHS_IN_YEAR;
        double monthlyInterestRate =
            yearlyInterestRate/MONTHS_IN_YEAR/100;
        double futureValue = 0;
        for (int i = 1; i <= months; i++)
            futureValue =
                (futureValue + monthlyPayment) *
                (1 + monthlyInterestRate);
        return futureValue;
    }
}
```

Object-Oriented Programming: Class

► Working with static fields:

The Product class with a static variable and a static method

```
public class Product {  
    private String code;  
    private String description;  
    private double price;  
  
    // declare a static variable  
    private static int objectCount = 0;  
    public Product() {  
        code = "";  
        description = "";  
        price = 0;  
        objectCount++;    // update the static variable  
    }  
    // get the static variable  
    public static int getObjectCount() {  
        return objectCount;  
    }  
    . . .  
}
```

Object-Oriented Programming: Class

► Syntax for calling a static field or method

```
className.FINAL_FIELD_NAME  
className.fieldName  
className.methodName(argumentList)
```

How to call static fields

From the Java API

```
Math.PI
```

From a user-defined class

```
FinancialCalculations.MONTHS_IN_YEAR  
Product.objectCount // if objectCount is  
                    // declared as public
```

Object-Oriented Programming: Class

► How to call static methods

From the Java API

```
NumberFormat currency =  
    NumberFormat.getCurrencyInstance();  
int quantity = Integer.parseInt(inputQuantity);  
double rSquared = Math.pow(r, 2);
```

From user-defined classes

```
double futureValue =  
    FinancialCalculations.calculateFutureValue(  
        monthlyPayment, yearlyInterestRate, years);  
int productCount = Product.getObjectCount();
```

A statement that calls a static field and a static method

```
// pi times r squared  
double area = Math.PI * Math.pow(r, 2);
```

Object-Oriented Programming: Class

- ▶ The syntax for coding a static initialization block

```
public class className
{
    // any field declarations

    static
    {
        // any initialization statements for static fields
    }

    // the rest of the class
}
```

Object-Oriented Programming: Class

► A class that uses a static initialization block

```
public class ProductDB
{
    // a static variable
    private static Connection connection;

    // the static initialization block
    static
    {
        try
        {
            String url =
                "jdbc:mysql://localhost:3306/MurachDB";
            String user = "root";
            String password = "sesame";
            connection = DriverManager.getConnection(
                url, user, password);
        }
    }
}
```

Object-Oriented Programming: Class

- ▶ A class that uses a static initialization block (cont.)

```
        catch (Exception e)
        {
            System.err.println(
                "Error connecting to database.");
        }
    }

    // static methods that use the Connection object
    public static Product get(String code){}
    public static boolean add(Product product){}
    public static boolean update(Product product){}
    public static boolean delete(String code){}
}
```

- ▶ Static initialization block is loaded when any of the static methods is called
- ▶ Use it where a static variable can't be initialized in the declaration itself, specially when it needs complex coding

Object-Oriented Programming: Class

- ▶ Can you process a static variable from a non-static method? Or vice versa?
- ▶ What is the purpose of static members (variable, and methods) anyways?
- ▶ What are the differences of instance members and static members
- ▶ Can you have a class:
 - ▶ with all static members,
 - ▶ without static members, or
 - ▶ with combination of members which are static and non-static?
- ▶ Static members make a program more efficient or less? Why, or why not?
- ▶ What are the gotchas of having static members?

Object-Oriented Programming: Class

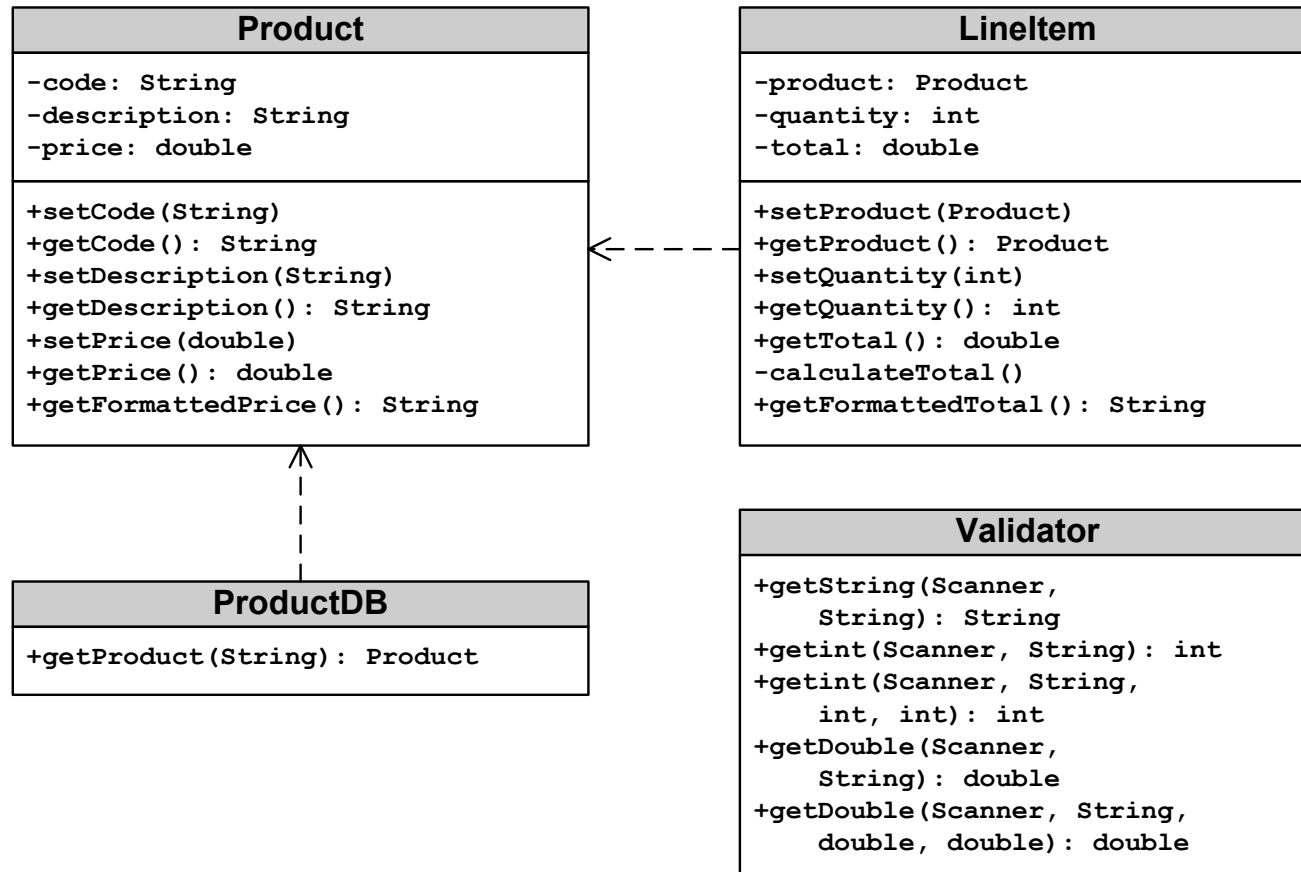
- ▶ null value: if a data field of reference type does not reference an object then the data field holds a value *null*
- ▶ Default value of data field (instance variable):
 - ▶ Reference type: null value
 - ▶ Primitive:
 - ▶ Numeric type: 0 value
 - ▶ boolean type: false
 - ▶ char: '\u0000'
- ▶ Default value of data field (local variable)
 - ▶ No default values
 - ▶ Error flagged by compiler if used without initializing

Object-Oriented Programming: Class

- ▶ **Immutable Objects and Classes:**
 - ▶ If the contents of an object cannot be changed after object is created, the object is called *immutable object* and its class is called *immutable class*
- ▶ **How do you create an immutable class?**
 - ▶ Create a class with all private members
 - ▶ Provide no mutators (methods which modify data) on these members
 - ▶ Additionally make sure you have no accessor methods for a reference which has mutable data field
 - ▶ Example: say, you are returning a Date reference from a immutable class without mutators, however, Date class is mutable
 - This will make your class also mutable, although your class itself has no mutators and all members are private

Object-Oriented Programming: Class

► The class diagrams for the Line Item application



Object-Oriented Programming: Class

► Code walk through LineItem application

```
Welcome to the Line Item Calculator

Enter product code: java
Enter quantity:      2

LINE ITEM
Code:                java
Description: Murach's Beginning Java
Price:               $49.50
Quantity:            2
Total:               $99.00

Continue? (y/n) :
```

Java API: More practice with Strings

► String Exercise

Example#1: What will be printed?

```
String str1 = "Bineet";
String str2 = "Bin";
String str3 = "Bineeta";
String str4 = "Bipins";
String str5 = "Bimal";
String str6 = "Bz";
String str7 = "Ba";

System.out.println(str1.compareTo(str2)); //? : 3
System.out.println(str1.compareTo(str3)); // -1
System.out.println(str1.compareTo(str4)); // -2
System.out.println(str1.compareTo(str5)); // 1
System.out.println(str1.compareTo(str6)); // -17
System.out.println(str1.compareTo(str7)); // 8
System.out.println(str1.compareTo(str1)); // 0
```

Java API:

► String Exercise (Cont.)

Example#2:

```
String s1 = "Adam";
String s2 = s1;
if (s1 == s2)    // ??
    System.out.println("s1: " + s1 + " s2: " + s2
                        + " is =");
else
    System.out.println("s1: " + s1 + " str2: " +
                        s2 + " is not=");
```

Example#3:

```
String s3 = "Elizabeth";
String s4 = "Elizabeth";
if (s3 == s4)    // ??
    System.out.println("s3: " + s3 + " s4: " + s4
                        + " is =");
else
    System.out.println("s3: " + s3 + " s4: " + s4
                        + " is not=");
```

Java API:

► String Exercise (Cont.)

Example#4:

```
String s3 = "Elizabeth", s4 = "Elizabeth";
s3 = "Elizabeth";          s4 = "Elizabeth";
if (s3 == s4)              // ??
    System.out.println("s3: " + s3 + " s4: " + s4 + "
                        is =");
else
    System.out.println("s3: " + s3 + " s4: " + s4 + "
                        is not=");
```

Example#5:

```
String s5 = new String("Elizabeth");
String s6 = new String("Elizabeth");
//s5="Elizabeth"; //what happens if we did this?
//s6="Elizabeth";
if (s5 == s6)              // ??
    System.out.println("s5: " + s5 + " s6: " + s6 + "
                        is =");
else
    System.out.println("s5: " + s5 + " s6: " + s6 + "
                        is not=");
```


Java API:

► String Exercise (Cont.)

Example#6:

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter s1: ");
String s1 = sc.next();
System.out.print("Enter s2: ");
String s2 = sc.next();

if (s1 == s2)    // ?? - case 1, different string,
                //case 2 same string
    System.out.println("s1: " + s1 + " s2: " + s2 + "
                        is =");
else
    System.out.println("s1: " + s1 + " s2: " + s2 + "
                        is not=");
```

Java API:

► String Exercise (Cont.)

Example#7:

```
String s1 = "OldString", s2 = "OldString";

if (s1 == s2)    // ??
    System.out.println("s1: " + s1 + " s2: " + s2 + "
                        is =");
else
    System.out.println("s1: " + s1 + " s2: " + s2 + "
                        is not=");

System.out.print("Enter s1: ");
s1 = sc.next();    //sc is scanner
System.out.print("Enter s2: ");
s2 = sc.next();

if (s1 == s2)    //Say you wrote "OldString" for both
    System.out.println("s1: " + s1 + " s2: " + s2 + "
                        is =");
else
    System.out.println("s1: " + s1 + " s2: " + s2 + "
                        is not=");
```

Java API:

► String Exercise (Cont.)

Example#8:

```
String s1 = "OldString";
String s2 = s1.toUpperCase();
if (s1 == s2)    // ??
    System.out.println("is =");
else if (s1.equals(s2))
    System.out.println(" is equal");
else
    System.out.println(" is not equal");
```

Example#9:

```
String s3 = "Hello ";
String s4 = s3;
s3 += "welcome to the class!";
//?? What s3 holds, and what s4 holds? And why?
if (s3 == s4)    // ??
    System.out.println(" is =");
else if (s3.equals(s4))
    System.out.println(" is equal");
else
    System.out.println(" is not equal");
```

Further Reading

- ▶ **Murach's Java Programming:**
 - ▶ Chapter 7, & 13

Next Lecture

- ▶ Object Oriented Programming
 - ▶ Inheritance
 - ▶ Interface
 - ▶ Polymorphism

What Will You Do This Week & Next?

- ▶ Go through lecture#3 material and demo codes
- ▶ Complete test your understanding quiz
 - ▶ Test your understanding quizzes are not graded
- ▶ Complete HW#3
 - ▶ It is available in the assignment section. Submit your work and you will be awarded with “Model Solution” right away.
 - ▶ Homework are not graded either
- ▶ Read ahead for lecture#4

Summary

- ▶ **Object Oriented Programming Concepts**
 - ▶ Thinking in objects
 - ▶ Abstraction
 - ▶ Encapsulation
 - ▶ Class and Object
 - ▶ UML diagram
 - ▶ Writing classes
- ▶ **String classes:**
 - ▶ Example