Mahidol University
Wisdom of the Land

Faculty of Information
and Communication Technology

LECTURE 03
DESIGNING CLASSES

ITCS209 Object Oriented Programming

Dr. Siripen Pongpaichet

(Some materials in the lecture are done by
Dr. Suppawong TuarobAnd Dr. Petch Sajjacholapunt)

28 January 2019

Semester 2/2018

Objectives

- After finish this class, student can
  - **Identify** each part in the Java class
  - **Design** Java classes to solve a given problem using OOP paradigm
  - **Write** Java classes to solve a given problem correctly

Outline

- Reflect Week 1 & 2 (+ Basic Java)
- Object References
- Implicit Parameters
- Designing Classes
- Accessors, Mutators, and Immutable Classes
- Static Methods
- Static fields
- Variable Scope
- Preconditions and Postconditions

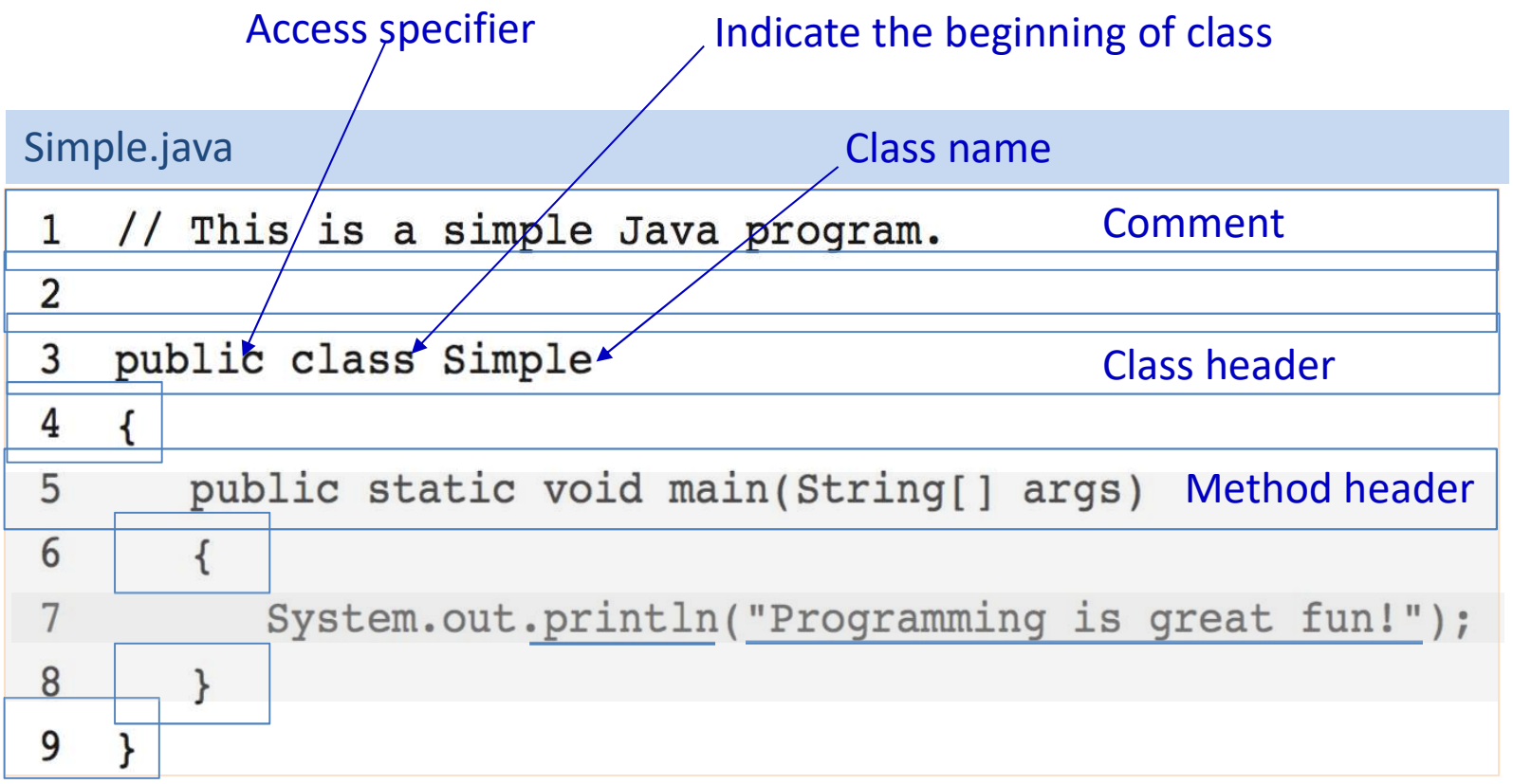
The Parts of java program

Simple.java

```
1  // This is a simple Java program.
2
3  public class Simple
4  {
5     public static void main(String[] args)
6     {
7        System.out.println("Programming is great fun!");
8     }
9  }
```

Class name | Comment | Class header | Method header

The Parts of java program



Everything between the two braces { } is called BODY

Java Key Words/ Reserved Words

abstract	const	for	int	public	throw
assert	continue	final	interface	return	throws
boolean	default	finally	long	short	transient
break	do	float	native	static	true
byte	double	goto	new	strictfp	try
case	else	if	null	super	void
catch	enum	implements	package	switch	volatile
char	extends	import	private	synchronized	while
class	false	instanceof	protected	this	

Java Special Characters

Characters	Name	Meaning
//	Double slash	Marks the beginning of a comment
( )	Opening and closing parentheses	Used in a method header
{ }	Opening and closing braces	Encloses a group of statements, such as the contents of a class or a method
" "	Quotation marks	Encloses a string of characters, such as a message that is to be printed on the screen
;	Semicolon	Marks the end of a complete programming statement

Working with Variables

Variable.java

```
1  // This program has a variable.
2
3  public class Variable
4  {
5     public static void main(String[] args)
6     {
7        int value;
8
9        value = 5;
10       System.out.print("The value is ");
11       System.out.println(value);
12    }
13 }
```

Pass Variable to the method

Variable Declaration

Assignment Statement

Program Output

What is the output????

Mahidol University
Faculty of Information
and Communication Technology

Data Type vs Object

- If we want to store **student's name**, what data type should we use?
- If we want to store **student's age** information, what data type should we use?
- How about If we want to store **car's** information, what data type should we use?

Implementing Class

Car.java

```
public class Car {
```

A

```
    private String brand;
    private String model;
    public double price;
```

B

```
    public Car(String carBrand, String carModel, double carPrice){
        brand = carBrand;
        model = carModel;
        price = carPrice;
    }
```

C

```
    public void setModel(double carModel){
        model = carModel;
    }
```

D

```
    public double getModel(){
        return model;
    }
```

E

```
    public void updateBrand(String carBrand, String carModel){
        brand = carBrand;
        model = carModel;
    }
```

F

```
    public String getCarInfo(){
        return "Car[brand=" + brand + ", model=" + model + ", price=" + price + "]";
    }
}
```

Identify each part in the program

Please match each term with each part of the program

- Attributes
- Constructor
- Getter method
- Setter method
- Method with no return
- Method with return

A

B

C

D

E

F

Implementing Class Tester

Declaration – variable name and object type
Instantiation – 'new' keyword
Initialization – call to a constructor

```
public class CarTester {

    public static void main(String[] args) {

        Car myCar = new Car("Toyota","Corolla Altis",970000);

        myCar.setModel("Camry");
        System.out.println(myCar.getModel());

        myCar.price = 880900;
        System.out.println(myCar.price);

        myCar.updateBrand("Mercedes-Benz", "C350e");
        myCar.price = 3280900;
        System.out.println(myCar.getCarInfo());

    }

}
```

Checkpoint

True/False

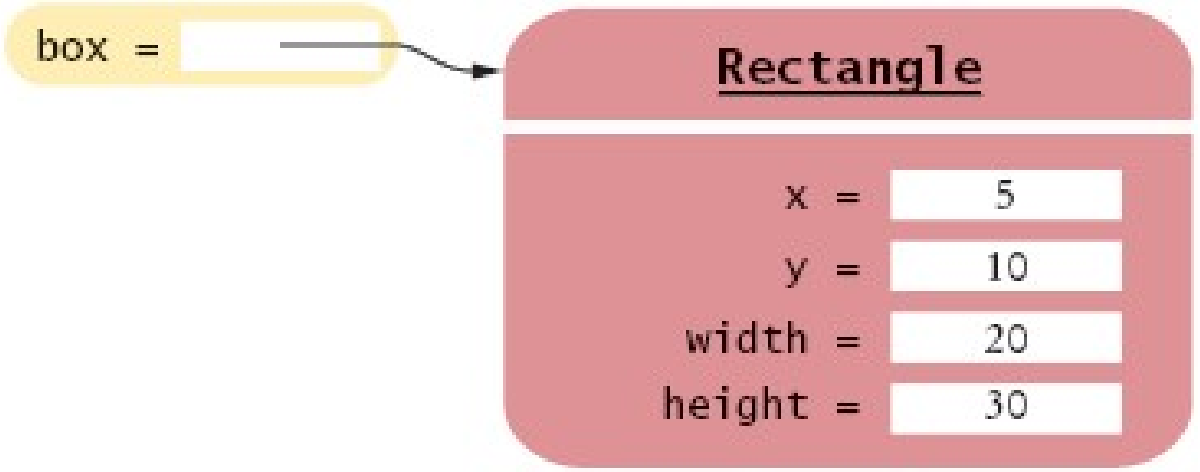
1. Java is case-sensitive language (e.g., `Main` ≠ `main`)
2. Every Java application program must have a method named `main`
3. Every *.java* file must have a method named `main`
4. You MUST write a statement in one line.
5. A class name MUST start with UPPERCASE
6. The data stored in a variable may change while the program is running (hence the name '*variable*')
7. Variables do not have to be declared before they can be used

Outline

- Recall Week 1 & 2 (+ Basic Java Syntax)
- Object References
- Explicit vs Implicit Parameters
- Designing Classes
- Accessors, Mutators, and Immutable Classes
- Static Methods
- Static fields
- Variable Scope
- Preconditions and Postconditions

Object references

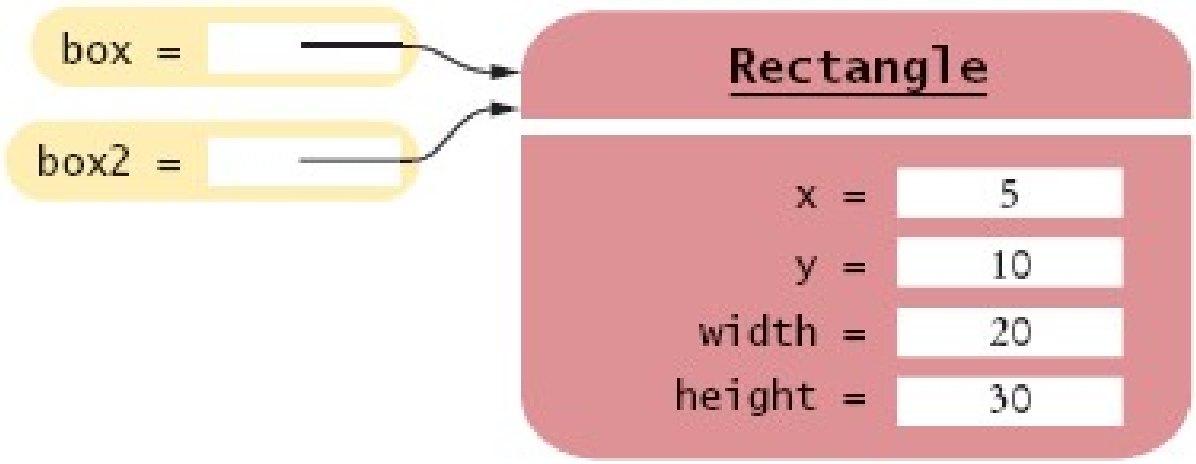
- Object reference describes the location of an object in memory
- The new operator returns a reference to a new object
- ```
  Rectangle box = new Rectangle(5, 10, 20, 30);
  ```


  ```

Object references (Cont)

- Multiple object variables can refer to the same object

```
Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box;
```



Notice

Changing value of variables in object ***box*** will affect value of variable in object ***box2***

Object references (Cont)

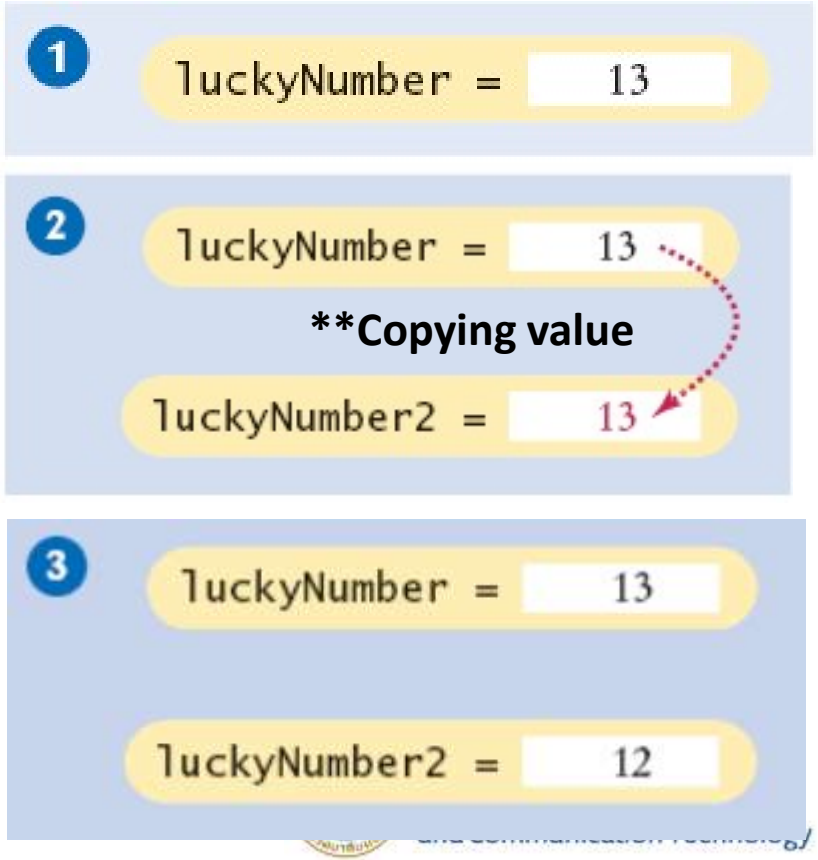
Primitive type variables ≠ **Object** variables (reference)

(e.g., int, double, boolean, …) (e.g., Student, Book, Car, BankAccount, …)

```
int luckyNumber = 13;
```

```
int luckyNumber2 = luckyNumber;
```

```
int luckyNumber2 = 12;
```



Outline

- Recall Week 1 & 2 (+ Basic Java Syntax)
- Object References
- Explicit vs Implicit Parameters
- Designing Classes
- Accessors, Mutators, and Immutable Classes
- Static Methods
- Static fields
- Variable Scope
- Preconditions and Postconditions

Parameters


```
public class Car{
  String color;
  int price;
  public Car(String c, int p){
    color = c;
    price = p;
  }

  pubic void setColor(String c){
    color = c;
  }
}
```

Car

color = white
price = 220000

setColor("black")

c = black
color = ~~white~~ black

```
public class CarTester{
   public static void main(String[] args){
    Car myCar = new Car("white",220000);
    myCar.setColor("black");
   }
}
```

Explicit Parameter

Implicit Parameter

Mahidol University
Faculty of Information and Communication Technology

Explicit Parameters

- Recall that a parameter is a value that is given to a method as input.
- Methods can have one or more parameters.

- TWO different kinds of parameters:
  - Explicit parameter: that is passed by specifying the parameter in the parenthesis of a method call.

```
System.out.println("Java is FUN!");
int price = 100;
System.out.println("The price is " + price);
myCar.updateBrand("Mercedes-Benz", "C350e");
```

Implicit Parameters

- Implicit parameter: the object on which the method is invoked (object reference before the name of a method).

```
public class Car{
  String color;
  int price;
  pubic void setColor(String c){
    color = c;
  }
}
```

```
Car myCar = new Car();
myCar.setColor("white");

Box myBox = new Box();
myBox.getArea();
```

```
public class Box{
  double width, height;
  …
  pubic double getArea(){
    return width * height;
  }
}
```

Implicit Parameters


```
class BankAccount{
    double balance;
    …
    public void withdraw(double amount) {
        double newBalance = balance - amount;
        balance = newBalance;
    }
}
```

`balance` is the balance of the object to the left of the dot

```
momsSavings.withdraw(500);
```

Compiler will translate the above statement to

```
double newBalance = momsSavings.balance – amount;
momsSavings.balance = newBalance;
```

Implicit Parameters and this

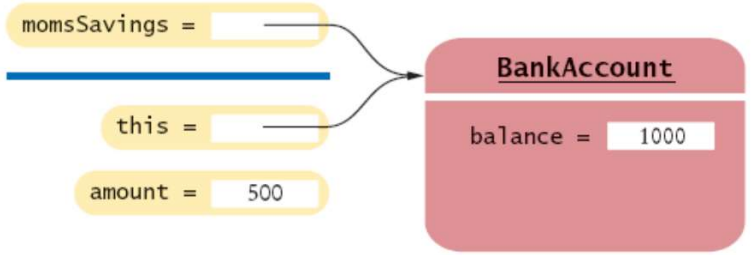
- The this reference denotes the implicit parameter

```
public void deposit (double amount) {
    balance = balance + amount;
}
```

```
public void deposit (double amount) {
    this.balance = this.balance + amount;
}
```

```
momsSavings.deposit(500);
```

**Java assumes that a variable name that are not an explicit *parameter* or a *local* variable must refer to an instance variable. That's why you don't need to write this before the variable balance



- You MUST use this to refer to an *instance variable* **when** you have the same variable name for *explicit parameter* or *local variable*.

```
public void setBalance(double balance){
    balance = balance;
}
```

NOT a syntax error but a logical error

```
public void setBalance(double balance){
    this.balance = balance;
}
```

Primitive Type Variables as Parameters

```
BankAccount harrysChecking = new BankAccount();
harrysChecking.balance = 2500;
double savingsBalance = 1000;
harrysChecking.transfer(500, savingsBalance);
System.out.println(savingsBalance);

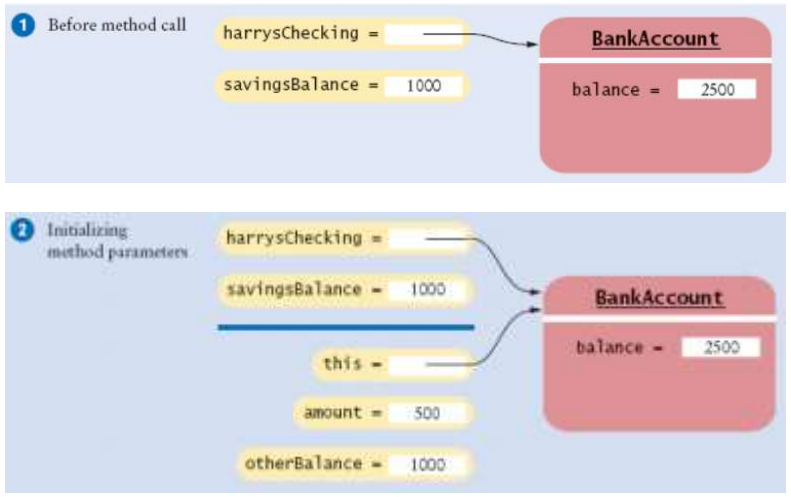
...
void transfer(double amount, double otherBalance) {
    balance = balance - amount;
    otherBalance = otherBalance + amount;
}
```

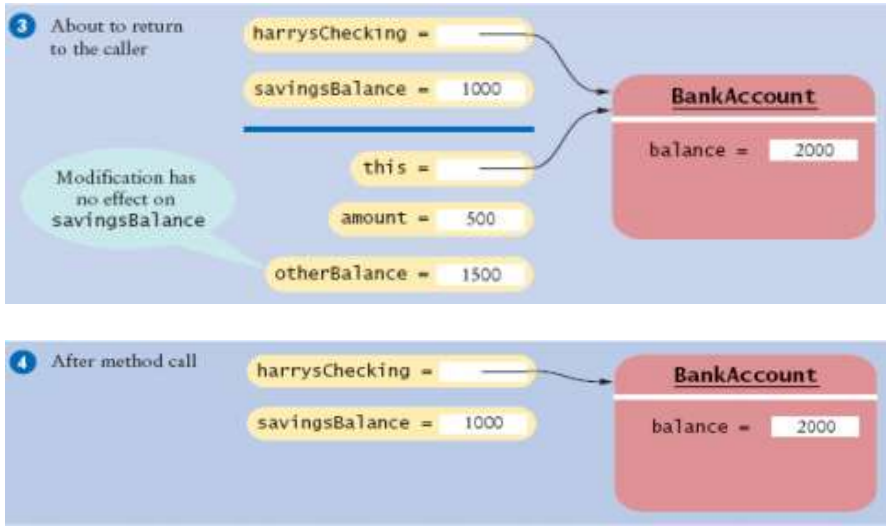
1 2 4

This code is in the `main` method

3

This method is in the `BankAccount` Class





Object References Variables as Parameters

```
public class BankAccount {

  private double balance;

  public BankAccount(double bal){
    balance = bal;
  }

  public void deposit(double bal){
    balance = balance + bal;
  }

  public void withdraw(double bal){
    balance = balance - bal;
  }

  public double getBalance(){
    return balance;
  }
}
```

```
public class ManageAccount {
   public static void main(String[] args){
   BankAccount myAccount = new BankAccount(1000);
   BankAccount myMomAcc = new BankAccount(20000);
   transfer(myMomAcc, myAccount, 2000);
   System.out.println("current balance in my mom
account: " + myMomAcc.getBalance());
   System.out.println("current balance in my
account: " + myAccount.getBalance());
   }
   public static void transfer(BankAccount source,
BankAccount target, double bal){
      source.withdraw(bal);
      target.deposit(bal);
   }
}
```

What is the output on your console?

Outline

- Recall Week 1 & 2 (+ Basic Java Syntax)
- Object References
- Explicit vs Implicit Parameters
- Designing Classes
- Accessors, Mutators, and Immutable Classes
- Static Methods
- Static fields
- Variable Scope
- Preconditions and Postconditions

Designing Classes

- A class represents a single concept from the problem domain
- Name for a class should be a noun that describes the concept
- Method names should be verbs
- Example concepts from mathematics:
  - `Point`
  - `Rectangle`
  - `Ellipse`
- Example concepts from real life:
  - `BankAccount`
  - `CashRegister`

Outline

- Designing classes
- UML class diagram
- Using multiple classes

Classes "describe" **objects**.
Classes are used to create objects.
Classes are a kind of factory – or blueprint – for *constructing* objects.
The non-static parts of the class describe what *variables* and *methods* the objects will contain.

Objects are instance of a **class**. → the process is called *instantiation*
Objects are created and destroyed as the program runs.
In one program, there can be **many objects** with the **same class.**

How do people design classes?

Steps to start…

1. Look for statements that identify objects in the problem statement. Write down the class name.

2. Look for statements that mention the attributes of the objects in the problem statement. Write down the attribute of the objects.

3. Look for statements that mention the behavior/process of the objects.

Example1 - bookstore

… Develop a program that assists bookstore employee.

For each book, the program should track

the book's title, its price, its year of publication,

and the author's name. The program should

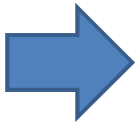
be able to update book information …

Example1 – bookstore (cont.)

- **Class**
  - Book
- **Attributes**
  - Book's title
  - Book's price
  - Year of publication
  - Author's name
- **Method**
  - Update

Example1 – bookstore (cont.)

- **Class**
  - Book
- **Attributes**
  - Book's title
  - Book's price
  - Year of publication
  - Author's name



```
class Book {
    String bookTitle;
    int bookPrice;
    int year;
    String authorName;
}
```

Example1 – bookstore (cont.)

Best practice for setting default variable is by using Constructor

Book.java

```
class Book {
   String bookTitle;
   int bookPrice;
   int year;
   String authorName;

   Book(String bTitle, int bPrice, int bYear, String bAuthor ){
    bookTitle  = bTitle;
    bookPrice  = bPrice;
    bookYear   = bYear;
    authorName = bAuthor;
   }
}
```

Example1 – bookstore (cont.)

To test the class Book, we should implement the class *BookTester* with main method (This is called 'Client Class').

BookTester.java

```
class BookTester {

   public static void main(String[] args) {
        Book b1 = new Book("Java book", 150, 2014, "Peter");
   }

}
```

Example1 – bookstore (cont.)

We can define and use value of a variable in class Book directly (if they are not defined as a private)

BookTester.java

```
class BookTester {

   public static void main(String[] args) {
      Book b1 = new Book("Java book", 150, 2014, "Peter");
      b1.authorName ="Patty";
      System.out.println(b1.authorName); //printing author name
   }
}
```

Example1 – bookstore (cont.)

- For consistency reason, when employee update the book information, they should update all field together.

- For example, sometime employee may forget to update some information such as they may update title of a book, price, and year but not author name.

Example1 – bookstore (cont.)

To prevent this from happening, we first need to limit the access of relevant attribute. Then design method to update all field at the same time.

Book.java

```
class Book {

    private String bookTitle;
    private int bookPrice;
    private int bookYear;
    private String authorName;

    Book(String bTitle, int bPrice, int bYear, String bAuthor ){
     bookTitle  = bTitle;
     bookPrice  = bPrice;
     bookYear   = bYear;
     authorName = bAuthor;
    }

    void updateBook(String bTitle, int bPrice, int bYear, String bAuthor){
     bookTitle  = bTitle;
     bookPrice  = bPrice;
     bookYear   = bYear;
     authorName = bAuthor;
    }
}
```

Example1 – bookstore (cont.)

Now employee cannot use the variable directly, they have to update variable by using the method `updateBook`.

BookTester.java

```
class BookTester {
   public static void main(String[] args) {
        Book b1 = new Book("Java book", 150, 2014, "Peter");
      b1.updateBook("C Book", 180, 2013, "Somsak");
   }
}
```

Designing class using UML

- UML is a pictures/diagram of an OO system
  - Programming languages are not abstract enough for OO design
  - UML is an open standard; lots of companies use it

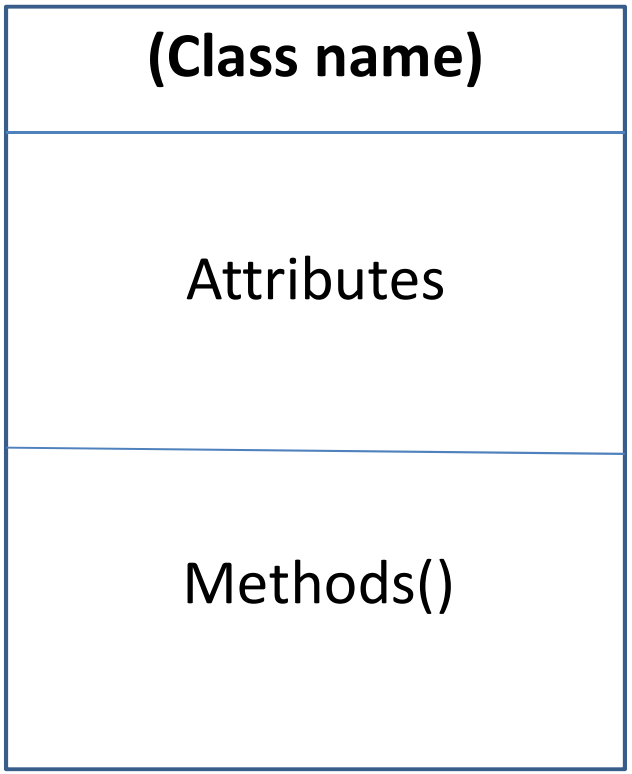
- Union of all Modeling Languages
  - Use case diagrams
  - Class diagrams
  - Object diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Statechart diagrams
  - Activity diagrams
  - Component diagrams
  - Deployment diagrams

UML class diagrams

- **UML class diagram** is a picture of:
  - the classes in an OO system
  - their fields and methods
  - connections between the classes

Basic Diagram of one class

- **Class** name in top of box
- **Attributes** (optional)
  - should include all fields of the object
- **Methods** (optional)



Class attributes (= fields)

- attributes (fields, instance variables)
  - ***Format*** *name* : *type* [*count*] = *default_value*
  - visibility: + public
    # protected
    - private
    ~ package (default)
  - underline static attributes
  - attribute example:

    - balance : double = 0.00

    + bookList : Book [0…*]

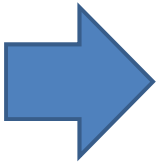
Class operations / methods

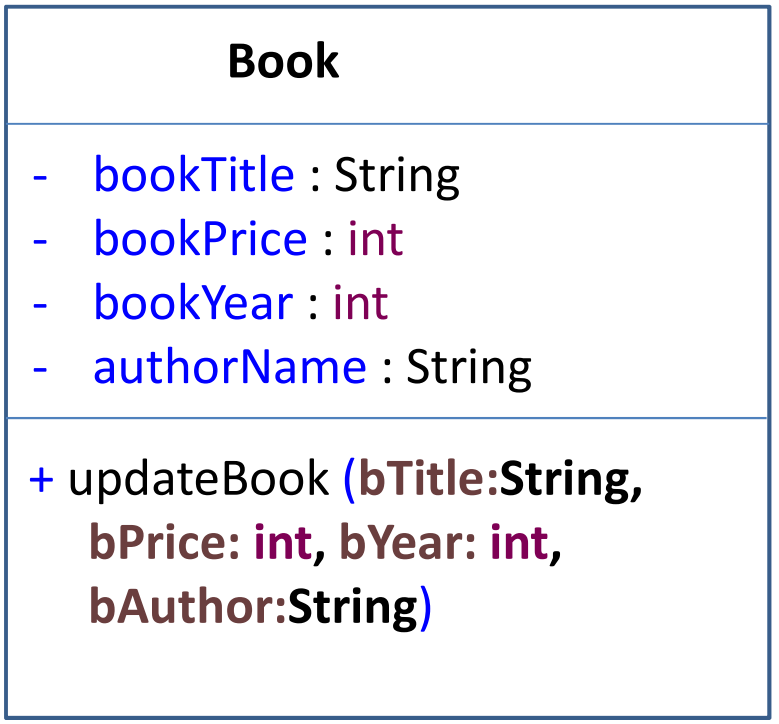
- operations / methods
  - ***Format*** *name* (*parameters*) : *return_type*
  - visibility: + public
    # protected
    - private
    ~ package (default)
  - underline static methods
  - parameter types listed as (name: type)
  - omit *return_type* on constructors and when return type is void
  - method example:

    + distance(p1: Point, p2: Point): double
    + printInfo()

UML for bookstore (cont.)

- **Class**
  - Book
- **Attributes**
  - Book's title
  - Book's price
  - Year of publication
  - Author's name
- **Method**
  - Update





Exercise (1)

… Implement a class Product. A product has a name and a price. Supply methods getName, getPrice, and reducePrice …

For testing (later in the lab)
Implement a program ProductPrinter that makes two products, prints the name and price, reduces their prices by 5.00%, and then prints the prices again.

Exercise (2)

… Implement a class Student. For the purpose of this exercise, a student has a name and a total quiz score. Supply an appropriate constructor and methods getName(), addQuiz(int score), getTotalScore(), and getAverageScore() …

Notice

To compute the average score, you also need to store the number of quizzes that the student took.

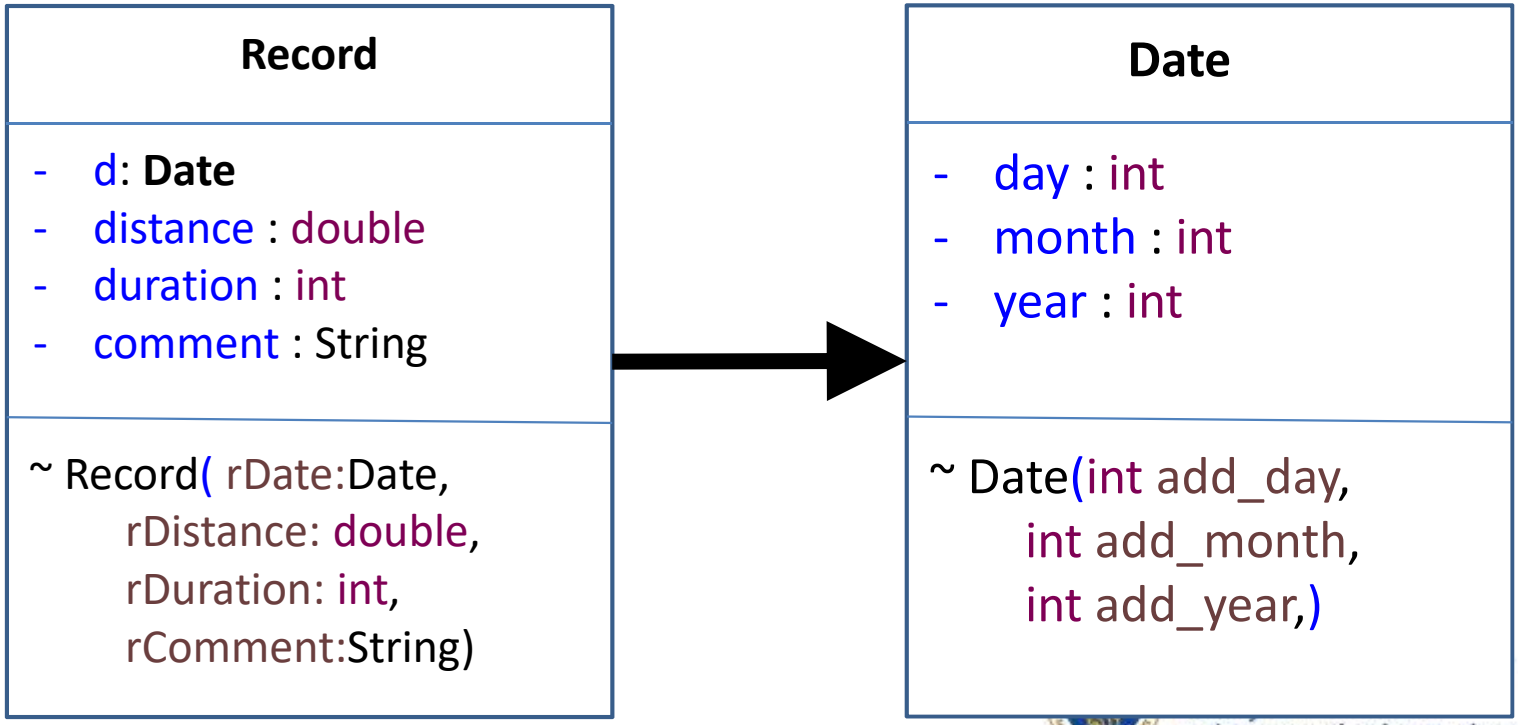
Multiple classes usage

- Sometime you need to deal with a problem that need more than one class. For example:

… Develop a program that manages a runner's training log. Every day the runner enters one record about the day's run. Each record includes the day's date, the distance of the day's run, the duration of the run, and a comment describing the runner's post-run …

Multiple classes usage

… Develop a program that manages a runner's training log. Every day the runner enters one record about the day's run. Each record includes the day's date, the distance of the day's run, the duration of the run, and a comment describing the runner's post-run …



Multiple classes usage (Cont)

Record.java

```
class Record {

     Date d;
     double distance;
     int duration;
     String comment;

     Record(Date rDate, double rDistance,
            int rDuration, String rComment)
     {
          d = rDate;
          distance = rDistance;
          duration = rDuration;
          comment = rComment;
     }

}
```

Date.java

```
class Date {

     int day;
     int month;
     int year;

     Date(int add_day,
          int add_month,
          int add_year)
     {
          day = add_day;
          month = add_month;
          year = add_year;
     }

}
```

RecordTester.java

```
public class RecordTester {
     public static void main(String[] args) {
          //Code here//
57   }
 }
```

Exercise (3)

… Develop a program that helps visitor navigate restaurant in Mahidol Salaya. The program must be able to provide four pieces of information for each restaurant: its name, the type of food it serves, its price range, and the address (street, district, postcode, phone number)…

- Example of data: (1) Mai-tok mai-tak, a Thai restaurant, inexpensive, on Phutthamonthon Sai 4 Salaya 73170 (0977797989).