Mahidol University
Faculty of Information
and Communication Technology

LECTURE 03
Objects and Class

ITCS123 Object Oriented Programming

Dr. Siripen Pongpaichet
Dr. Petch Sajjacholapunt
Asst. Prof. Dr. Ananta Srisuphab

Semester 2/2023 12 January 2024

Recap – Lecture 02

- Variable & Declaration*
- Data Type
  - Primitive
  - Reference
- Reading Input (Scanner Class)
- Decision (if, if-else, if-else-if, switch)
- Iteration (for, while, nested loop)

You can check your lab assignment score on MyCourses by selecting “Grade” menu

Note that each lab has 3 points

In the past week, how many **hours** you spent on coding outside the classroom?

*__Declaration__: a formal or explicit statement or announcement.

Announcement

- **Quiz 1**
  - Today in class (14:30 – 15:00)
  - Topics: Lecture 1 – 3
  - Multiple choices and short answers
  - **Closed-book**

Outcomes of this lecture

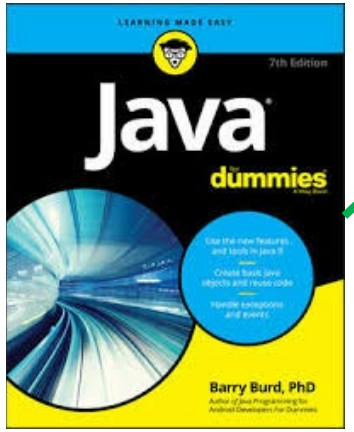
- Can *create* a Class and construct an Object
- Can *explain* and *create* constructor, setter and getter methods
- Can *explain* the scope of the variables (public, private, protected)
- Can *use* Objects in the program

OBJECTS AND CLASSES

How to represent books in a program

Multiple Associated Variables without Class

How to represent these two books?




```
public static void main (String[] args){

    String b1title = "Java Concepts";
    String b1ISBN = "1118423011";
    String b1author =  "Cay S. Horstmann";
    int b1edition = 7;
    int b1pages = 848;

    String b2title = "Java for Dummies";
    String b2ISBN = "1119235553";
    String b2author =  "Barry Burd";
    int b2edition = 7;
    int b2pages = 504;

    …

}
```

What will happen if I want to have **all books'** information in the library ?

1.1 Data items of Objects

- Each book has the following of data associated with them such as
  - Title
  - ISBN
  - Author(s)
  - Publisher
  - Edition
  - Number of pages
  - Etc.

- Most useful programs do not just manipulate numbers and strings. They deal with ***data items*** or ***properties*** that more closely represent real-word **objects**.

1.2 Behaviors of Objects

- To access or change values of data items of objects, OOP programs usually use "***methods***" of objects.
- These methods are resemble to the **behaviors** of real-world objects such as
  - Display book's information
  - Return number of authors
  - Update book's price when you have a discount
  - Etc.
- Each method consists of a **sequence of statements** that can access the internal data items of an object.
- Each method must clearly define: method's name, arguments (take any input?), return (produce an output?), what they do.

1.3 Instance Variables and Methods

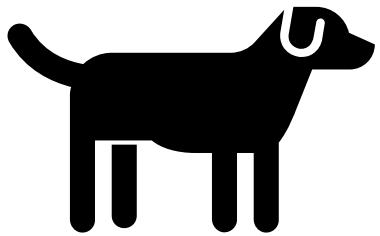
- An object consists of **data items** and **behaviors**.
  - **Data items** -> Instance Variables = Instance Fields = Attributes
  - **Behaviors** -> Methods = Function (in C programming)

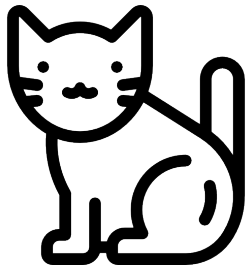
```java
public class SimpleBook {
    // Instance Variables (i.e., data items)

    private String title;
    private double price;

    // Methods (i.e., behaviors)

    public void printInfo() {
        System.out.println("Title: " + title +  " $" + price);
    }

    public void setPrice(double newPrice) {
        price = newPrice;
    }
}
```

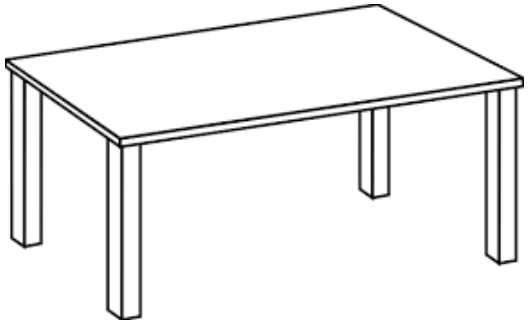
1.4 Classes: Modeling world with classes

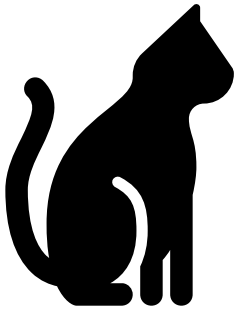
Can you group these 6 objects into classes?

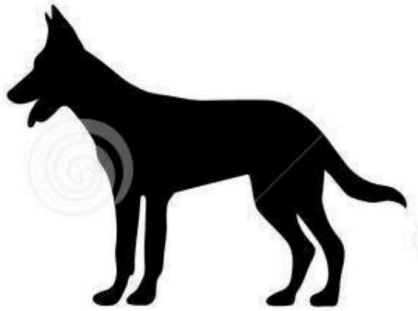
How many classes do you have?







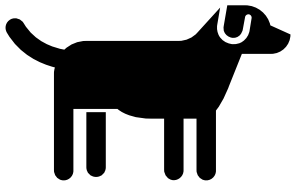


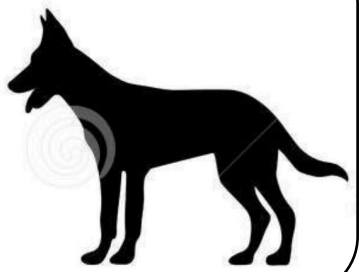


Example

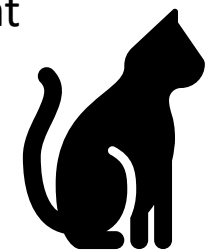
Animal

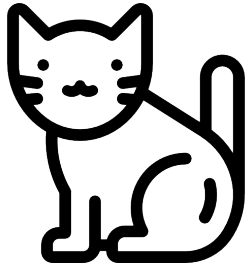
Dog





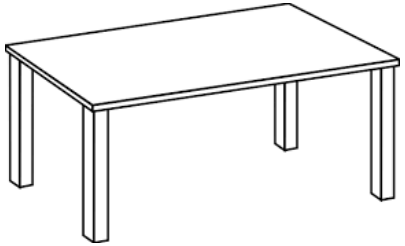
Cat





These classes are abstrations of the reality.

Table



We can have many other classes, depending on how broad we want the class to be.
We can come up with a class that fit all things together for example, “four-legged-things” class.

Classes

- Class is a **conceptual model** or an **abstraction** of reality.
- Class describes the commonalities of similar objects

```
• Book  -> Java Concepts, Java for Dummies, Python 101, …
• Dog   -> white dog, black dog, brown dog, dotted dog …
• Table -> rectangle table, round table, square table, …
```

Class

Object ~~ contains actual values

- Object is an **instance of** a class. In another word, a class is a blueprint of an object.
- Objects of the same class share the same kind of **properties** and **behavior**.

1.5 Using Objects

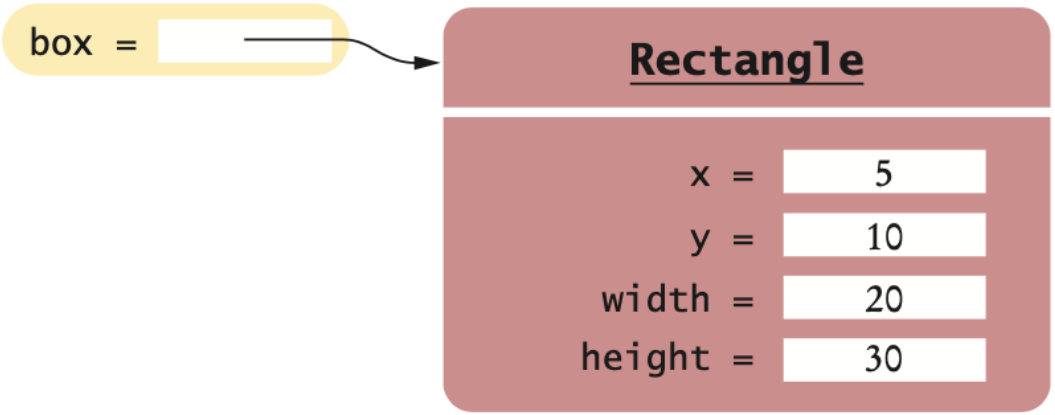
- In OOP paradigm, you will have to put **objects** together as its building blocks.
- Some objects are premade* and ready to use.
- But sometimes, you may need to design your own objects -> The design or blueprint of your objects is called a “class”
- To use an object, we need to **construct** an object from its class and **call the methods** defined by its class.

*Premade: prepared or made beforehand

CONSTRUCTING OBJECT

2.1 Reference Variable

- Object is a **reference variable**
- This reference variable contains the ***address*** to the block of memory that holds some data values.



- **Syntax:**

```
ClassName objectVar = new ClassName(arguments);
```

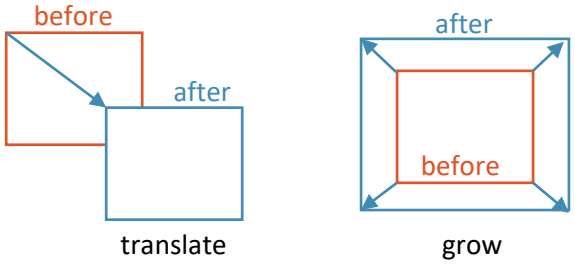
new operator -> creates an object and yields the object's address
arguments -> is a list of values needed for constructing objects

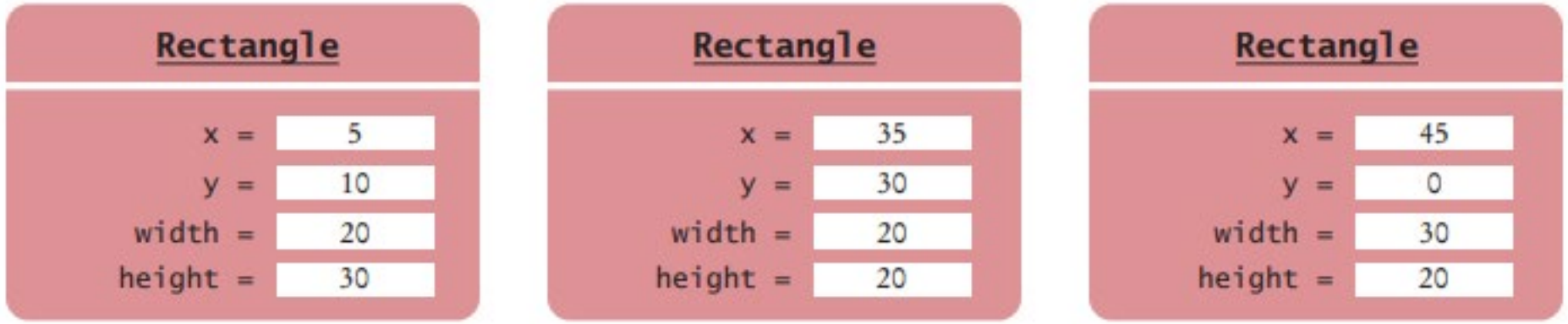
```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

Note that this object does not actually have a real rectangle shape in the memory. It just simply stores four numbers.

Example 1: Constructing Rectangle Object

- Class `Rectangle` (from `java.awt.Rectangle` package) describes the rectangle:
  - Instance variables (data items): `int x, int y, int width, int height`
  - Methods (behaviors): `.translate(int dx, int dy), .grow(int h, int v), ...`
- We can construct objects of Rectangle class that have different values in the instance variables.
  - Instance variables remain the same (`x, y, width, height`)
  - Values are different:





Explanation

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

1. Declare box variable (reference variable) with Rectangle class data type
2. The new operator creates a Rectangle object in memory
3. It uses the arguments (5, 10, 20, 30) to initialize the object's attributes
4. It returns the object (to be precise -> object's address)
5. The = (assignment) operator assigns the object's address to the variable box

```java
// 1
Rectangle a = new Rectangle(5, 10, 15, 20);

// 2
Rectangle b = (5, 10, 15, 20);

// 3
Rectangle c = Rectangle(5, 10, 15, 20);

// 4
Rectangle d;
d = new Rectangle(5, 10, 15, 20);

// 5
Rectangle e;
e.translate(15, 25);

// 6
Rectangle f = new Rectangle();
```

OR

Suggestion: You should check their API documents for more information.
(see next slide)

https://docs.oracle.com/javase/8/docs/api/java/awt/Rectangle.html

Java API Document: https://docs.oracle.com/javase/8/docs/api/

java.awt.Rectangle: https://docs.oracle.com/javase/8/docs/api/java/awt/Rectangle.html

Fields

Modifier and Type	Field and Description
int	**height** The height of the Rectangle.
int	**width** The width of the Rectangle.
int	**x** The X coordinate of the upper-left corner of the Rectangle.
int	**y** The Y coordinate of the upper-left corner of the Rectangle.

Constructors

Constructor and Description
Rectangle() Constructs a new Rectangle whose upper-left corner is at (0, 0) in the coordinate space, and whose width and height are both zero.
Rectangle(Dimension d) Constructs a new Rectangle whose top left corner is (0, 0) and whose width and height are specified by the Dimension argument.
Rectangle(int width, int height) Constructs a new Rectangle whose upper-left corner is at (0, 0) in the coordinate space, and whose width and height are specified by the arguments of the same name.
Rectangle(int x, int y, int width, int height) Constructs a new Rectangle whose upper-left corner is specified as (x,y) and whose width and height are specified by the arguments of the same name.
Rectangle(Point p) Constructs a new Rectangle whose upper-left corner is the specified Point, and whose width and height are both zero.
Rectangle(Point p, Dimension d) Constructs a new Rectangle whose upper-left corner is specified by the **Point** argument, and whose width and height are specified by the **Dimension** argument.
Rectangle(Rectangle r) Constructs a new Rectangle, initialized to match the values of the specified Rectangle.

2.2 Multiple Constructors

```java
import java.awt.Rectangle;

public class App {
    public static void main(String[] args) throws Exception {
        Rectangle box = new Rectangle(5, 10, 15, 20);
        Rectangle box2 = new Rectangle();
        System.out.println(box);
        System.out.println(box2);
    }
}
```

OUTPUT

```
java.awt.Rectangle[x=5,y=10,width=15,height=20]
java.awt.Rectangle[x=0,y=0,width=0,height=0]
```

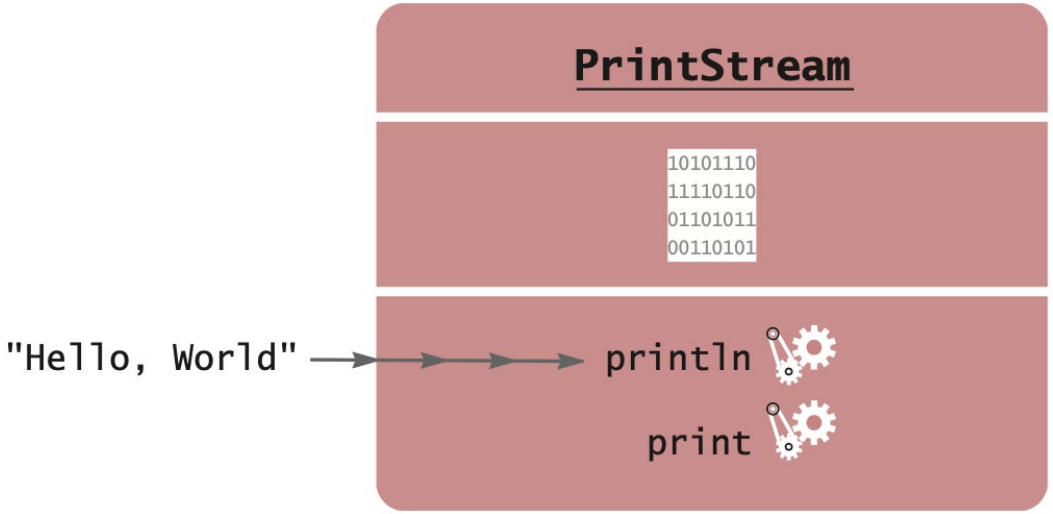
CALLING METHOD

3.1 Arguments & Return Values

- Method without return values

```
String greeting = "Hello, World!";
System.out.println(greeting)
```

1. Object: `System.out`
   This object is from the `PrintStream` class.
2. Method: `println(...)`
3. Argument: `greeting`.
4. Return: `void` (return nothing).



3.1 Arguments & Return Values

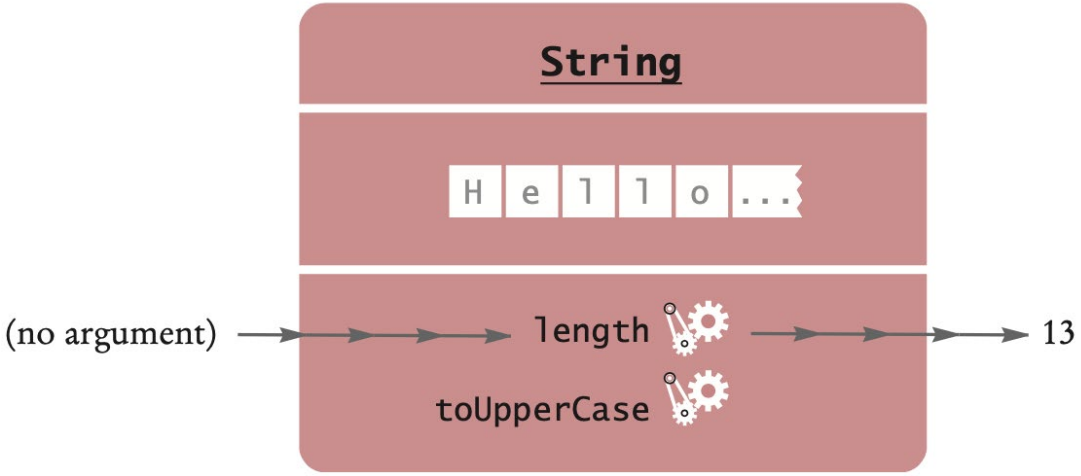
- Method without arguments

```
String greeting = "Hello, World!";
int numChars = greeting.length();
```

Note. You can skip **new** expression in constructing a String object. The String class has a special **shortcut** for its construction.

```
String str = "abc";

// This is equivalent to
String str1 = new String("abc");
```

1. Object: `greeting`
   This object is from the `String` class.
2. Method: `length(...)`
3. Argument: None.
4. Return: `int`, the number of characters.

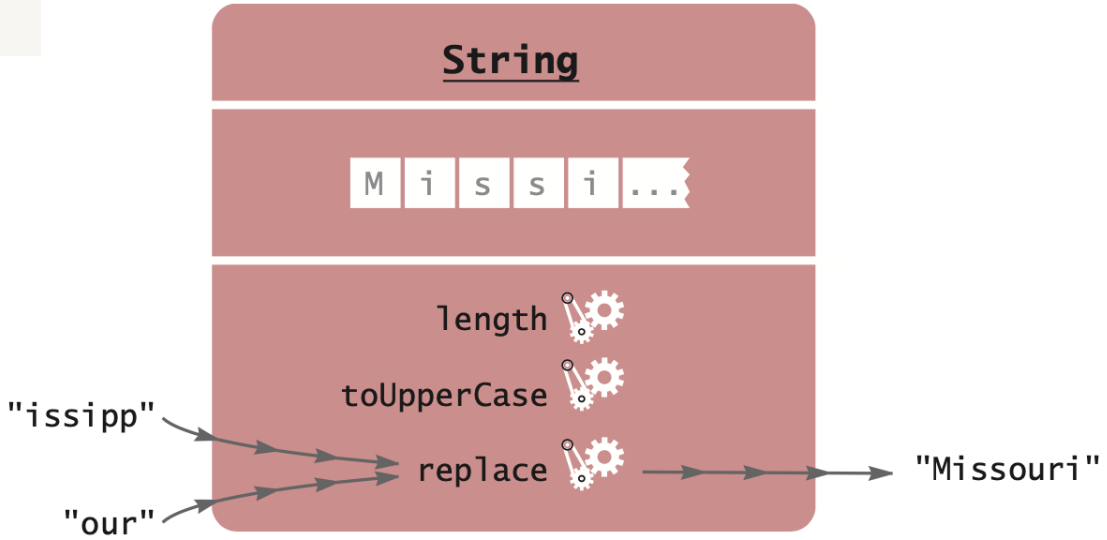


3.1 Arguments & Return Values

- Method with one or more arguments

```
String river = "Mississippi";
river = river.replace("issipp", "our");
```

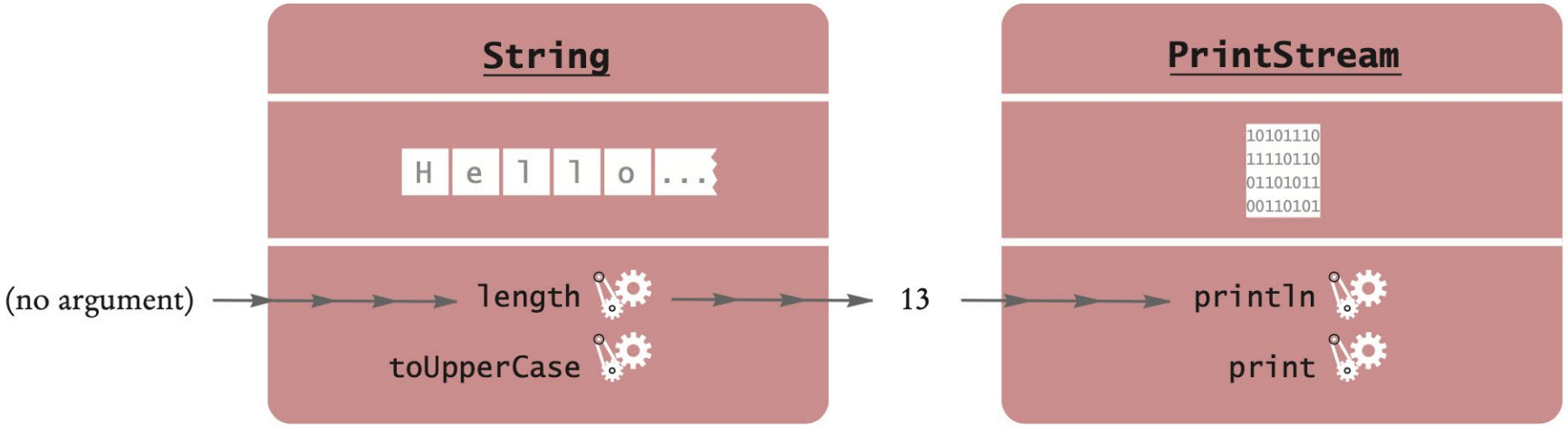
1. Object: `river`
   This object is from the `String` class.
2. Method: `replace(...)`
3. Arguments: `String` and `String`
4. Return: `String`



3.1 Arguments & Return Values

- **Nested Method:** The inner methods are evaluated and the returned value is passed as an argument of the outer methods.

```
int numberOfCharacters = greeting.length();
System.out.println(greeting.length());
```



3.2 Accessor Methods

- A method that accesses an object and returns some information about it, **without** changing the object, is called an **accessor** method.
- For example,
  - `length()` method of the String class returns a number of characters of the String object but does not change any data.
  - `getX()`, `getY()`, `getWidth()`, and `getHeight()` methods of the Rectangle class also just returns their information.
  - `getTitle()`, and `printBookInfo()` methods of Book class are also accessor methods

Example Code

```
double width1 = box.getWidth();
double width2 = box.getWidth();
System.out.println(width1 == width2);
/* Output:
true
*/
```

```
String greeting = "Hello";
System.out.println(greeting.toUpperCase());
System.out.println(greeting);
/* Output:
HELLO
Hello
*/
```

- Note that “toUpperCase()” method returns a text in uppercase **without** changing the original string value. In fact, all methods of String are accessor methods. Thus, we can say that String object is **immutable***

*Immutable: unchanging over time or unable to be changed.

3.3 Mutator Methods

- A method whose purpose is to **change** the internal data of an object is called a **mutator** method.
- For example,
  - `translate(int dx, int dy)` method of the Rectangle class is used to move the location of rectangle by changing the x and y values of the Rectangle object.
  - `setPrice(double price)` methods of Book class is used to set a new price of the Book object.

translate

```
public void translate(int dx,
                      int dy)
```

Translates this Rectangle the indicated distance, to the right along the X coordinate axis, and downward along the Y coordinate axis.

Parameters:

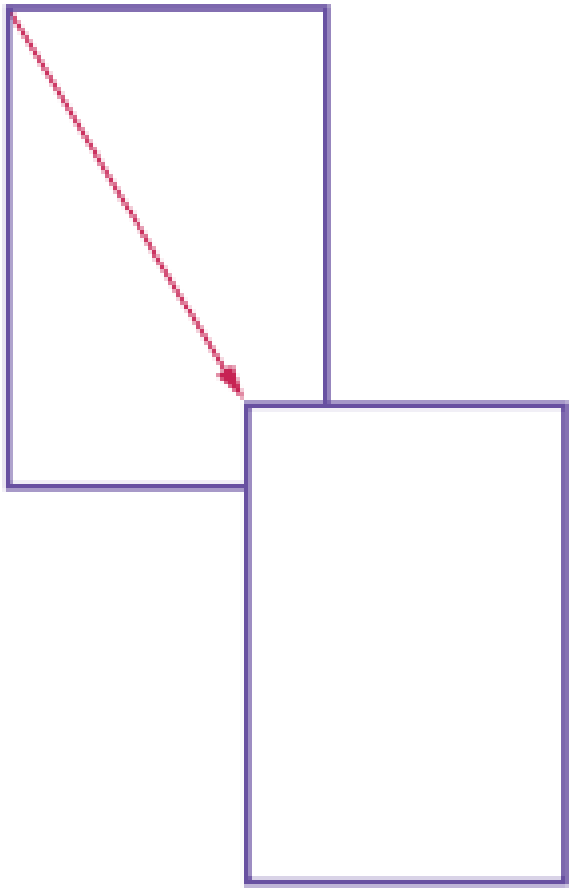
dx - the distance to move this Rectangle along the X axis

dy - the distance to move this Rectangle along the Y axis

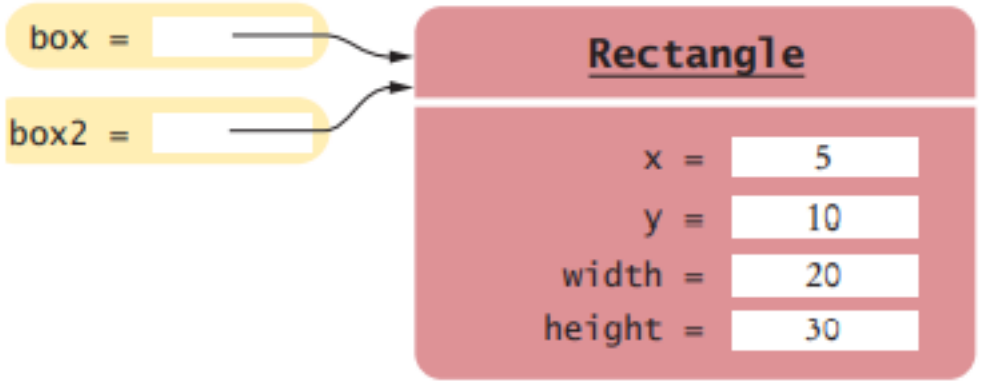
See Also:

setLocation(int, int), setLocation(java.awt.Point)

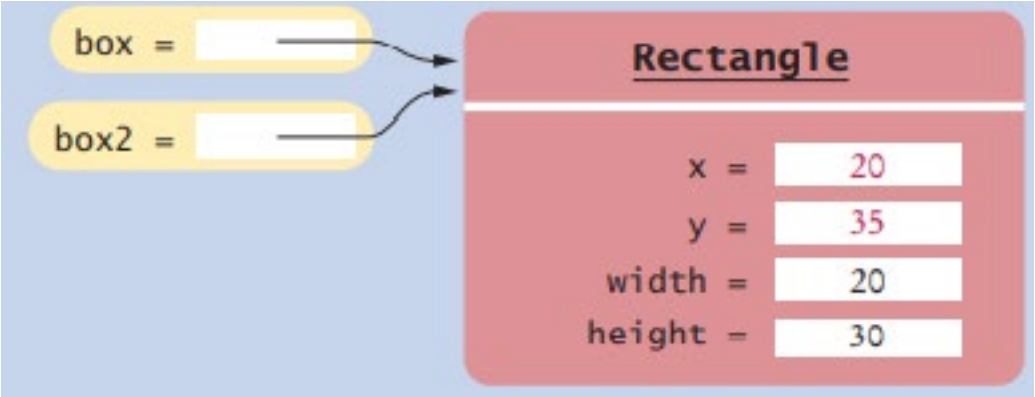
```
Rectangle box = new Rectangle(5, 10, 15, 20);
System.out.println(box);
box.translate(5, 10);
System.out.println(box);
/* Output:
java.awt.Rectangle[x=5,y=10,width=15,height=20]
java.awt.Rectangle[x=10,y=20,width=15,height=20]
*/
```



Object References ~ Be Careful




```
Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box;
box2.translate(15, 25);
System.out.println(box);
System.out.println(box2);

/* Output:
java.awt.Rectangle[x=20,y=35,width=20,height=30]
java.awt.Rectangle[x=20,y=35,width=20,height=30]
*/
```



IMPLEMENTING CLASS

To design your own classes, construct an object from your class, and call its methods

4.1 Instance Variables or Attributes

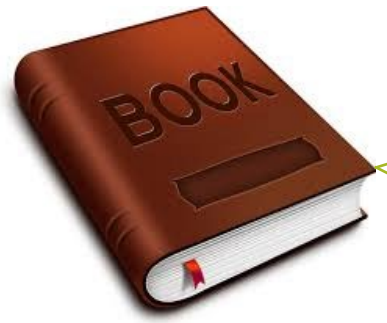
- **Instance Variables** are variables declared in a class.
- They will hold the value of data items of an object constructing from the class

```
//-------------- Syntax ----------------
public class ClassName {
  private typeName variableName;
   // . . .
}
//-----------------------------------
```

```
public class Book {

    private String title;
    private double price;

}
```

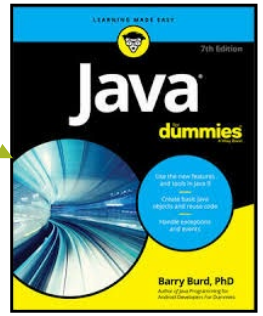
4.1.1 Instance Variables of Object

- Each object has its own set of instance variables. Different objects constructing from the **same class** will have the **same set of instance variables**, but these instance variables can store **different values**.



Class book is a blueprint of object
There are two instance variables:
title and **price** in this class


```
title = "Java Concepts"
price = 120.50
```




```
title = "Java for dummies"
price = 69.99
```

4.2 Constructor Method

- **Constructor** is a special method that has the **same name as the class**.
- It is normally used to set initial value of attributes of the object.
- An object is constructed via one of its constructors.

```
//-------------- Syntax ----------------
public class ClassName {
  public ClassName(arguments) {
    // body . . .
  }
}
//-----------------------------------
```

```
public class Book {

    private String title;

    public Book(String t){
        title = t;
    }

}
```

4.2.1 Multiple Constructors

```
public class Book {

    private String title;
    private double price;

    public Book(String t){
        title = t;
        price = 0.0;
    }

    public Book (String t, double p){
        title = t;
        price = p;
    }

}
```

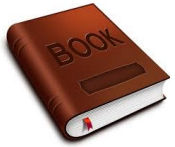
4.3 Constructing Object

Explanation

1. The **new** operator will first create the object in the memory.

2. Then the constructor -> **`Book(String t)` OR `Book(String t, double p)`** is invoked.

```
public class Book {

    public String title;
    public double price;

    public Book(String t){
      title = t;
      price = 0.0;
    }
    . . .
}
```



CLASS

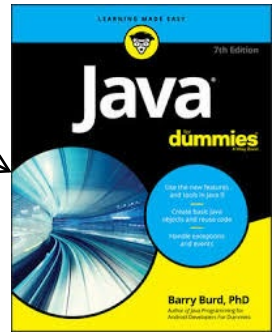
(Example)

```
Book book1 = new Book("Java Concepts");
```

Instance Objects

What is the value of price of book1?

```
Book book2 = new Book("Java for Dummies", 69.99);
```



Instance Objects

ConstrutingObje | Book.java | Example.java | Object.class

```
1
2  public class Book {
3
4      public ! private itle;
5      public ( private ice;
6
7      public Book(String t) {
8          title = t;
9          price = 0.0;
10     }
11
12     public Book(String t, double p) {
13         title = t;
14         price = p;
15     }
16
17     public static void main(String[] args) {
18         Book book1 = new Book("Java Concepts");
19         Book book2 = new Book("Java for Dummies", 69.99);
20
21     }
22
23 }
24
```

Variab | Break | Expre

Name	Value
no metho...	
this	Book (id=22)
price	0.0
title	"Java Concepts" (id=23)
t	"Java Concepts" (id=23)

<Choose a previously entered expression>

4.3 Other Types of Methods

- We implement all methods in a **class**, but we use methods from an **object**.
- Each object will have their **own instance variables** whose values can be modified via mutator methods

- Instance methods
  - Getter (accessor method)
  - Setter (mutator method)
  - Without/With return value
- Static methods

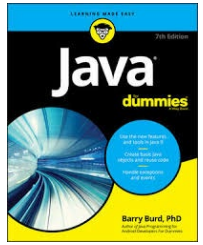
Getter Methods (Accessor Methods)

```
public class Book {

    private String title;

    public Book(String t){
     title = t;
    }

    public String getTitle(){
      return title;
    }

    public void setTitle(String t){
       title = title;
    }

}
```

General Methods
(return value **String**)

Normally, a common method can be called by
objectName.methodName()

```
public static void main(String[] args) {

Book book1 = new Book("Java Concepts");

System.out.println(book1.getTitle());

// return: Java Concepts

Book book2 = new Book("Java for Dummies");

System.out.println(book2.getTitle());

// return: Java for Dummies

}
```



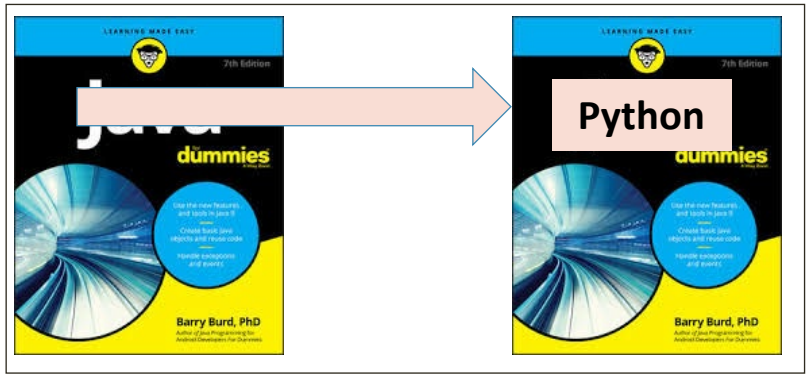
Setter Methods (Mutator Methods)

```
public class Book {

    private String title;

    public Book(String t){
        title = t;
    }

    public String getTitle(){
      return title;
    }

    public void setTitle(String x){
        title = x;
    }
}
```

Keyword void is used for non return value method

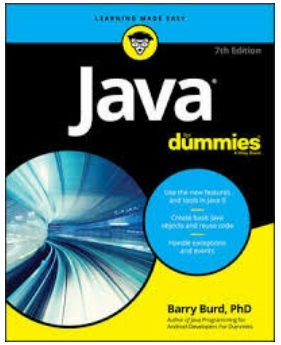
Parameter
(An input variable that is used in method)

```
public static void main(String[] args) {

    Book book2 = new Book("Java for Dummies");

    book2.setTitle("Python for dummies");

    System.out.println(book2.getTitle());

    // return: Python for Dummies

}
```



Instance Methods with no return (void)

```
public class Book {
    private String title;

    public Book(String t){
     title = t;
    }

    public void printBookInfo(){
      System.out.println(title);
    }

    public void printShortBook(int x){
        System.out.println(title.substring(0,x));
    }

    public static void main(String[] args) {
         Book book2 = new Book("Java for Dummies");
         book2.printBookInfo();
         // print: Java for Dummies
         book2.printShortBook(4);
         // print: Java
    }
}
```



OUTPUT

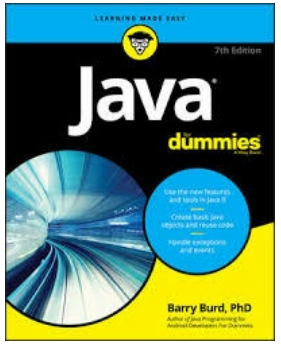
Java for Dummies
Java

Instance Methods with return

```
public class Book {
    private String title;
    private double price;

    public Book(String t, double p){
     title = t;
     price = p;
     }

    public double fullPrice (){
     return price;
     }

    public double discountPrice(int percent){
     return price - (percent/100.0*price);
     }

    public static void main(String[] args) {
        Book book2 = new Book("Java for Dummies", 200.00);
        System.out.println("Full price: " + book2.fullPrice());
        // print -> Full price: 200.00
        System.out.println("Discount price: " + book2.discountPrice(10));
        // print -> Discount price: 180.00
    }
}
```

OUTPUT

Full price: 200.0
Discount price: 180.0



Static Methods

- A "static" keyword can be used with method, variable, and class.
- A static method is a method that can be called without creating an instance of object.
- It belongs to a class instead of a specific object of class.

```
public class Book {

    public static void explain(){
        System.out.println("This is a book class.");
    }

    public static void main(String[] args) {
        Book.explain();

        int x = Math.abs(-5);
        System.out.println(x);
    }
}
```

OUTPUT

```
This is a book class.
5
```

4.4 How do you execute/run a Class?

- The class code *usually* has **NO main method**
- error 'no executable' found

Book.java **Class**

```
public class Book{
    String title;
    String ISBN;
    String author;
    …

    public book (…)
    {…}

    public showDetail()
    {…}

    public setAuthor(…)
    {…}

    Public getPages() {…}
}
```

BookManager.java **Client**

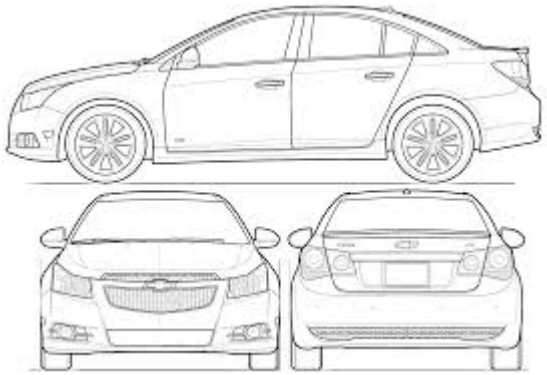
```
public class BookManager{
     public static void main (String[] args){
          Book book1 = new Book("Java Concepts",
          "1118423011", "Cay S. Horstmann", 7, 848);
          Book book2 = new Book("Java for Dummies",
          "1119235553", "Barry Burd", 7, 504);

          System.out.println("Book 1 Detail");
          book1.showDetail();
          book1.setAuthor("Siri Pon");

          int pageDiff = book1.getPages() - book2.getPages();

          System.out.println("page diff: " + pageDiff );
          …
     }
}
```

SCOPE OF VARIABLES
And
ENCAPSULATION

Another Example

Conceptual Model

CLASS



- **Attribute**
  - Color
  - Wheels
  - Distance
- **Behavior**
  - setColor()
  - getColor()
  - Run()

Objects

- **Attribute**
  - Color: **black**
  - Wheels: 4
  - Distance: 0
- **Behavior**
  - setColor()
  - getColor()
  - Run()

- **Attribute**
  - Color: **red**
  - Wheels: 4
  - Distance: 0
- **Behavior**
  - setColor()
  - getColor()
  - Run()

5.1 Type of variables

Instance variable : declared inside class but outside method.

Static variable: declared as static, all objects instantiated from this class share the same static variable.

```
public class Car {

    public String carColor = "black";
    public static int wheel = 3;

    public Car(String color){
     carColor = color;
    }

    public String getColor(){
      return carColor;
    }

    public void setColor(String color){
       String colorCode = color;
       carColor = colorCode ;
    }

}
```

This is useful when you want all objects from this class to share the global variable. For example, all car has the same number of wheel.

Another useful usage is to have a static **COUNT** variable keeping the value of number of objects instantiated from this class.

Local variable : declared inside method.

5.2 Usage of Static Variables

- To share a value among all objects of the same class (save memory)
  - i.e., faculty name and maximum GPA of ICT students class

```java
public class ICTStudent {

     public static String facultyName = "ICT";
     public static double maxGPA = 4.0;
     ...
}
```

```java
public static void main(String[] args) {
  ICTStudent first = new ICTStudent();
  ICTStudent second = new ICTStudent();
  System.out.println("Faculty: " + first.facultyName);
  System.out.println("Max GPA: " + second.maxGPA);
}
```

```
Faculty: ICT
Max GPA: 4.0
```

- Count how many objects have been created or generating a running number for an ID of an object

```java
public class ICTStudent {

  public static int latestID = 111;
  public int studentID;

  public ICTStudent() {
    latestID = latestID + 1;
    studentID = latestID;
    System.out.println("new object");
}
```

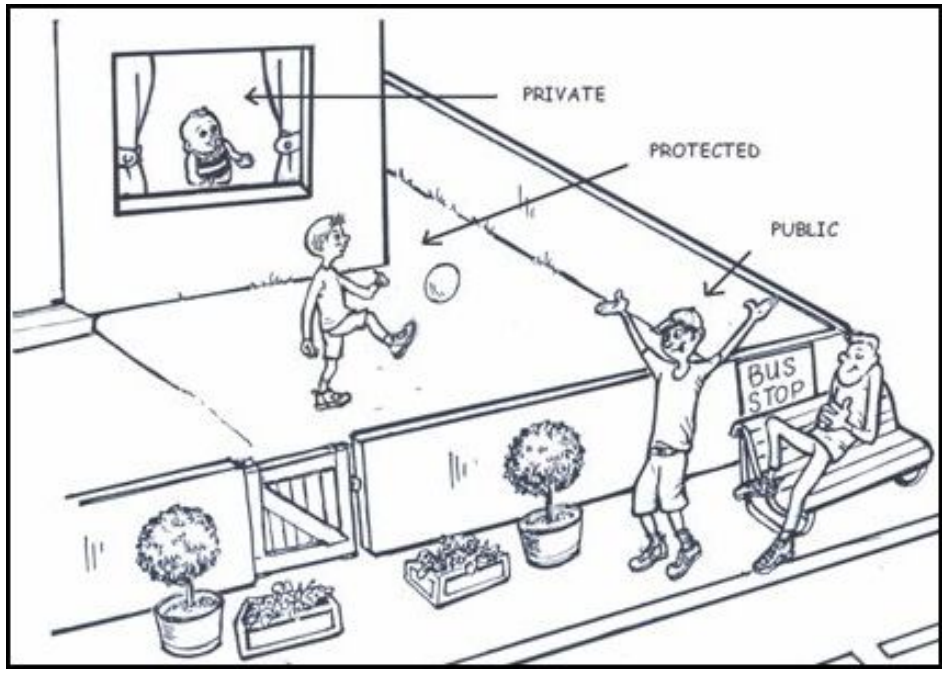
```java
public static void main(String[] args) {
  ICTStudent first = new ICTStudent();
  ICTStudent second = new ICTStudent();
  System.out.println("1st ID: " + first.studentID);
  System.out.println("2nd ID: " + second.studentID);
}
```

```
New Object
New Object
First ID: 112
Second ID: 113
```

5.3 Access specifiers

- Need to consider scope of variable

```
public String carColor = "black";
```



- Look at access specifiers in more detail:

Access Modifiers	Default	private ★	protected	public ★
Accessible inside the class	yes	yes	yes	yes
Accessible within the subclass inside the same package	yes	no	yes	yes
Accessible outside the package	no	no	no	yes
Accessible within the subclass outside the package	no	no	yes	yes

5.4 Encapsulation

- In the previous topic, we declare our instance variables using **`private`** **access specifier**
- With private access specifier, that instance variables can only be accessed by methods of the same class.

```java
// Book.java
public class Book {

    private String title;
    private double price;

    public Book(String t, double p) {
        title = t;
        price = p;
    }
    // . . .
    public String getTitle() {
        return title; // OK
    }
}
```

```java
// App.java
public class App {


    public static void main(String[] args) {
        Book myBook = new Book("Java", 100);
        String title = myBook.title; // ERROR
    }


}
```

Encapsulation

- **Encapsulation** is the process of **hiding implementation** details and providing public methods for data access.
- Sometime we need to hide information from being edited.

Read-only

"A student's name is given once when the object is created.
It should be visible to use but not changed."

- Sometime we need an attribute to be modified by only restricted channels to keep consistency. (provide a public method to modify their private instance variables)

"A student's address and phone number must be change together when student moves."

Encapsulation

```
public class Student{
    private String stdName;
    private String stdAddress;
    private String stdPhoneNumber;

    public Student(String name){
        stdName = name;
        stdAddress = "";
        stdPhoneNumber ="";
    }

    public void updateInfo(String
    address, String phone){
        stdAddress = address;
        stdPhoneNumber = phone;
    }

}
```

```
peter.stdName = "Sam";
peter.stdPhoneNumber = "0858185752";
```

```
public class ManageStudents{

  public static void main(String[] args){

    Student peter = new Student("Peter");

    peter.updateInfo("20 High Street Coventry CV2
3AH", "0858185752");

  }

}
```

Summary

- A class defines the variables and methods that an object of its type should have.
- An object (instance) is *instantiated* from a class. All variables and methods in an object must be implemented and ready to be called.
- private (class variables or methods) <=> Inside organs. Other people cannot see nor touch.
- public (class variables or methods) <=> Outside organs. Other people can see and modify.
- static (class variables or methods) <=> There exists only one copy in the entire program.

Object-Oriented Design for *'Course Enrollment'*

Object

Attributes

Methods

Student1
name: Phillip studentID: 1234 email: 1234@xyz.edu …
registerCourse(…) dropCourse(…) getGPA() …

Student2
name: Lisa studentID: 1235 email: 1235@xyz.edu …
registerCourse(…) dropCourse(…) getGPA() …

Course1
name: OOP courseID: ITCS209 Instructor:Teacher1 …
setInstructor(…) addStudent(…) removeStudent(…) getTotalStudents() …

Teacher1
name: Ing staffID: 71 email: 71@xyz.edu …
teachCourse(…) addCourseGrade(…) …