Chapter 7: Introduction to Classes and Objects

Java™ How to Program, 11th Edition

Instructor: Zhuozhao Li

Objectives

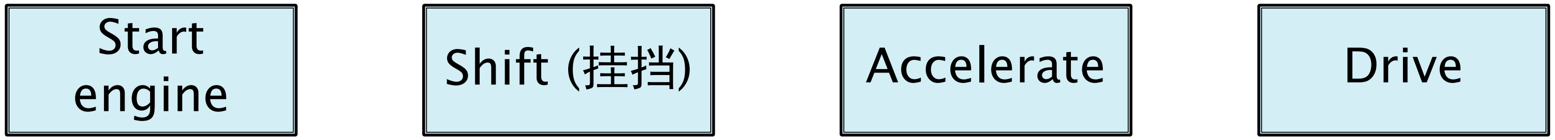
- Object-oriented Programming (面向对象编程，OOP)
- Understand classes (类), objects (对象), instance variables (实例变量)
- Learn to use a constructor to ensure that an object’s data is initialized when the object is created

Object-oriented Programming (面向对象编程，OOP)

- Typically, Java applications consist of one or more classes, each containing one or more methods.
  - E.g., Welcome1, Welcome2, Addition, Comparison…
- Each class (类) represents a type of objects (对象)
- Objects interact with each other for computations

Procedural Programing (面向过程编程, PP)

- a list of instructions to tell the computer what to do step-by-step
- E.g., Drive a car



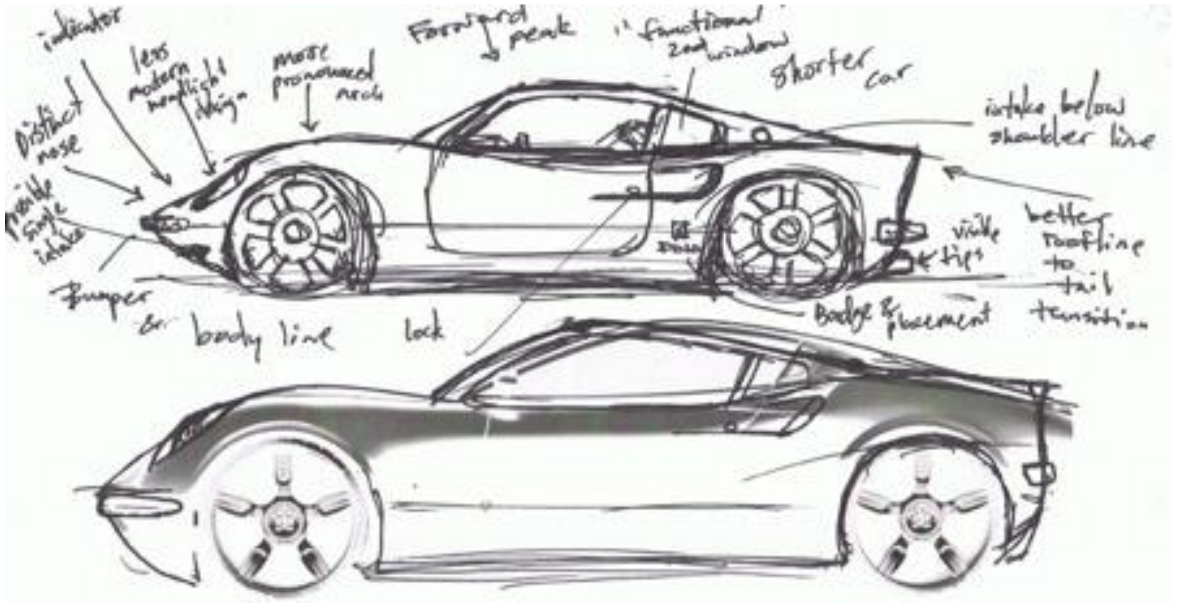
- A series of computational steps to be carried out
- FORTRAN, ALGOL, COBOL, BASIC, Pascal and C

Object-oriented Programming (面向对象编程，OOP)

- PP results in complicated code in real world applications
- E.g., make a car (including wheels, brake, pedals, etc.)
- Abstract to objects
- Closed to human thinking and real world. OOP is complementary (互补的) with PP
- Three key concepts: Class, Object, Method

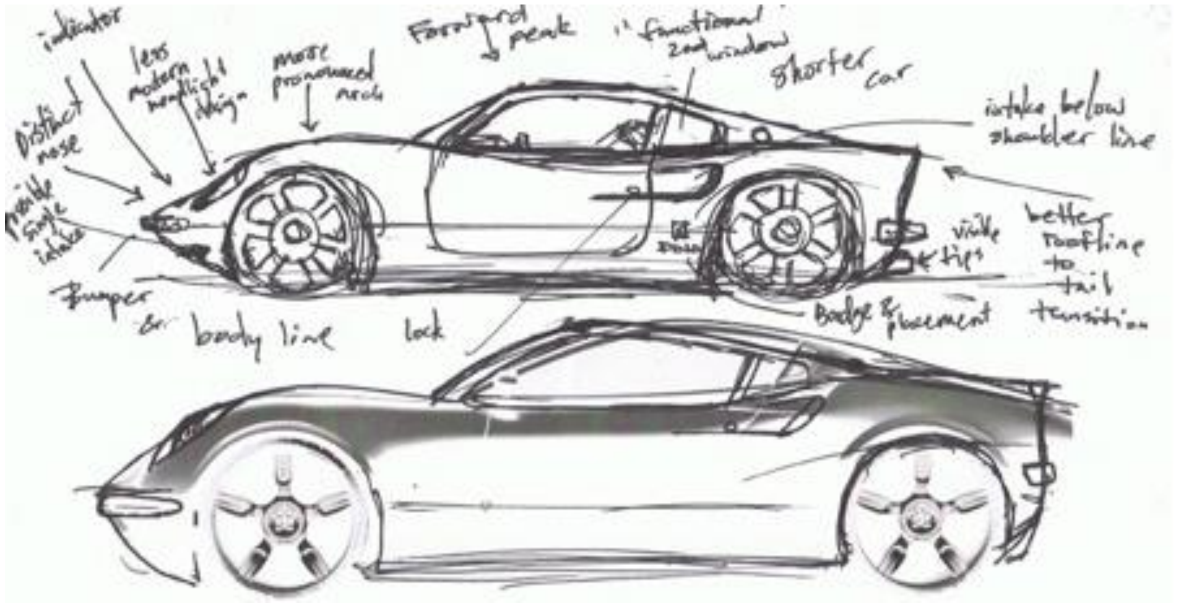
Example: Make and Drive a car

- Suppose we want to drive a car and accelerate it by pressing down on its accelerator pedal （加速踏板）



Classes, Objects, Methods

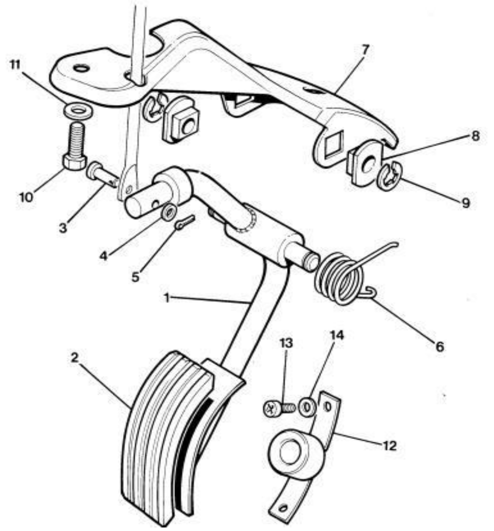
- Before you can drive a car, someone has to design it (engineering drawings/blueprints).



Classes, Objects, Methods

- Including the design for an accelerator pedal
- A lot more designs, e.g., the brake pedal, the steering wheel

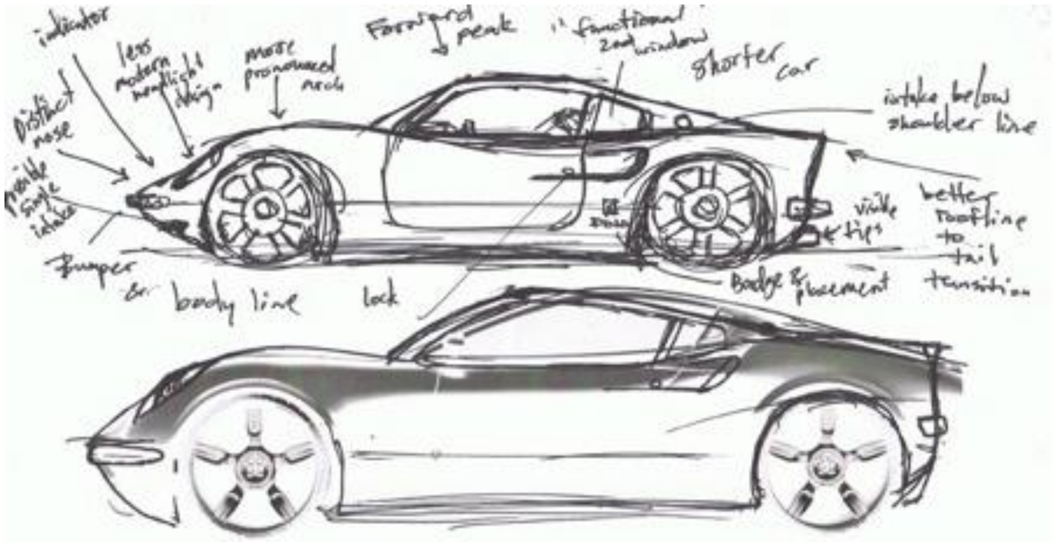
We don’t need to know the complex mechanisms behind the design to drive the car



Have you wondered the mechanisms to generate a random number in random.nextInt() method

Classes, Objects, Methods

- We cannot drive a car’s engineering drawings



Classes, Objects, Methods

- We cannot drive a car’s engineering drawings
  - Before we drive, it must be built from the engineering drawings
  - Even building a car is not enough, the driver must press the accelerator pedal to perform the task of drive the car

Classes, Objects, Methods

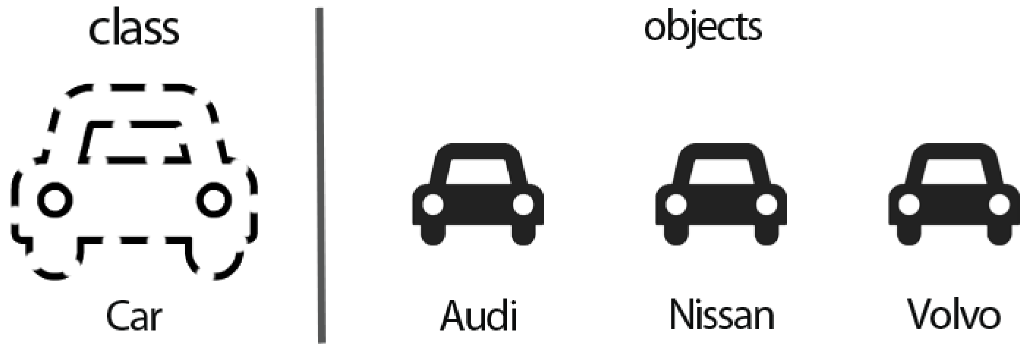
- Three key concepts in Java
  - class – a car’s engineering drawings (blueprint)
  - object – the car we drive
  - method – designed to perform tasks (make a car move)

Classes, Objects, Methods

- When programming Java, we begin by creating **a program unit called class**, just like we begin with engineering draws in the driving example.
- In a class, we provide one or more **methods** that are designed to perform the class's tasks.
- The method hides from its user the complex tasks that it performs, just like the accelerator pedal of a car hides from the driver the complex mechanisms that make the car move faster.

Classes, Objects, Methods

- We cannot drive a car's engineering drawings. Similarly, we cannot "drive" a class to perform a task
- Just as we have to build a car from its engineering drawings before driving it, we must build an **object** of a class before getting a program to perform tasks.



Classes, Objects, Methods

- When driving a car, pressing the accelerator pedal sends a message to the car to perform a task – make the car go faster.
- Similarly, we send a message to an object – implemented as a **method call** that tells a method of the object to perform its task.

Instance Variables

- A car can have many attributes (属性), such as its color, the amount of gas in its tank, its speed, and the total miles driven
- These attributes are represented as part of a car's design in its engineering diagrams
- As you drive a car, these attributes are always associated with the car (not other cars of the same model)
- Every car maintains its own attributes (e.g., knowing how much gas is left in its tank, but do not know about other cars)

Instance Variables (实例变量)

- Similarly, an object has attributes that are carried with the object as it's used in a program.
- These attributes are specified as the class's instance variables.
- E.g., a bank account object has a balance attribute (implemented as an instance variable) that represents the amount of money in that account.

The Whole Picture

- Class – a car’s engineering drawings (a blueprint)
- Method – designed to perform tasks (e.g., making a car move)
- Object – the car we drive
- Method call – perform the task (e.g., pressing the accelerator pedal)
- Instance variable – to specify the attributes (e.g., the amount of gas)

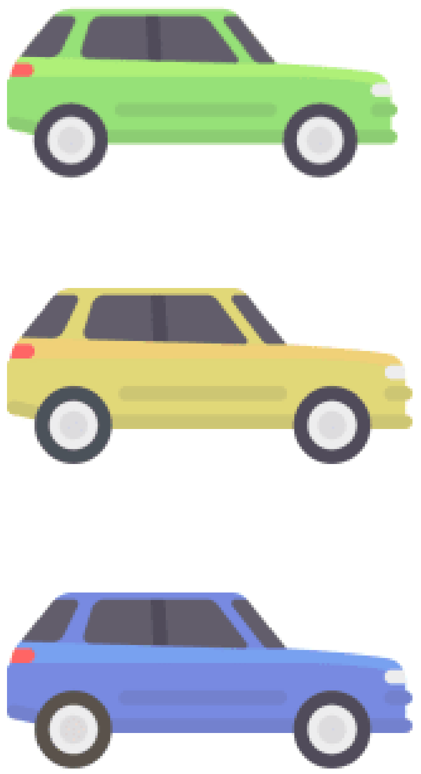
The Whole Picture

```
public class Car{
    private string _color;
    private string _model;
    private string _makeYear;
    private string _fuelType;

    public void Start(){
        ..
    }

    public void Stop(){
        ..
    }

    public void Accelerate(){
        ..
    }
}
```



Declaring a Class

Every class declaration contains the keyword `class` + the class' name

```
public class GradeBook {
   // every class' body is enclosed in a pair of
   // left and right curly braces
}
```

The **access modifier** `public` indicates that the declared class is visible to all classes everywhere.

Declaring a Method

A class usually consists of one or more methods.
Method = Method header + Method body (enclosed by {})

```
public class GradeBook {
   // display welcome message to the user
   public void displayMessage() {
      System.out.println("Welcome to the Grade Book!");
   }
}
```

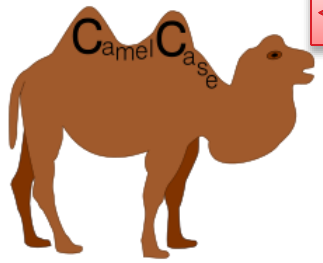
Declaring a Method Cont.

The **return type** specifies the type of data the method returns after performing its task, `void` means returning nothing to its calling method.

```
public class GradeBook {
   // display welcome message to the user
   public void displayMessage() {
      System.out.println("Welcome to the Grade Book!");
   }
}
```

The **access modifier** `public` indicates that the method is “available to public”, that is, can be called from the methods of other classes.

Declaring a Method Cont.



By convention, **method names** are in **Lower Camel Case**: the initial letter is in lower case, subsequent words begin with a capital letter.

```
public class GradeBook {
   // display welcome message to the user
   public void displayMessage() {
      System.out.println("Welcome to the Grade Book!");
   }
}
```

The parentheses enclose the information that the method requires to perform its task. Empty parentheses indicate no information needs.

Declaring a Method Cont.

Like class, the method body is also enclosed in {}. The method body contains **statements** that perform the method's task.

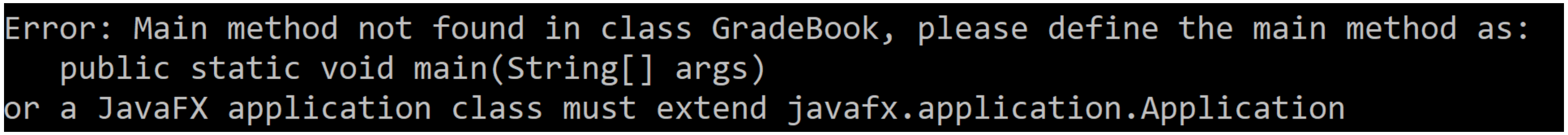
```java
public class GradeBook {
   // display welcome message to the user
   public void displayMessage() {
      System.out.println("Welcome to the Grade Book!");
   }
}
```

(1) Don't forget the ; after a statement; (2) try to use meaningful names when declaring a method to make your programs understandable.

Can We Run the Program?

What would happen if we run “`Java GradeBook`”?

```java
public class GradeBook {
   // display welcome message to the user
   public void displayMessage() {
      System.out.println("Welcome to the Grade Book!");
   }
}
```



```
Error: Main method not found in class GradeBook, please define the main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

Object Creation and Method Calling

```
public class GradeBookTest {
    public static void main(String[] args) {
        // create a GradeBook object
        // assign it to myGradeBook
        GradeBook myGradeBook = new GradeBook();

        // call myGradeBook's displayMessage method
        myGradeBook.displayMessage();
    }
}
```

Define a variable of the type GradeBook. Note that each new class you create becomes a new type, this is why Java is an **extensible language**.

Object Creation and Method Calling

```
public class GradeBookTest {
    public static void main(String[] args) {
        // create a GradeBook object
        // assign it to myGradeBook
        GradeBook myGradeBook = new GradeBook();

        // call myGradeBook's displayMessage method
        myGradeBook.displayMessage();
    }
}
```

Class instance creation expression. The keyword **new** is used to create a new object of the specified class. Class name + () represent a call to a **constructor** (a special method used to initialize the object's data).

Object Creation and Method Calling

```
public class GradeBookTest {
    public static void main(String[] args) {
        // create a GradeBook object
        // assign it to myGradeBook
        GradeBook myGradeBook = new GradeBook();

        // call myGradeBook's displayMessage method
        myGradeBook.displayMessage();
    }
}
```

We can use the variable `myGradeBook` to refer to the created object and call the method `displayMessage()` using the member operator ".". The empty parentheses indicate that we provide no additional data **(arguments)** to the called method.

Declaring a Method with a Parameter

- Sometimes a method needs additional information to perform its task. **Parameters** are for this purpose.

```
public class GradeBook {
   // display welcome message to the user
   public void displayMessage( String courseName ) {
      System.out.println("Welcome to the Grade Book for
      the course%s!\n“, courseName);
   }
}
```

Method Call with Arguments

```
public class GradeBookTest {
    public static void main(String[] args) {
        GradeBook myGradeBook = new GradeBook();
        myGradeBook.displayMessage("Java Programming");
    }
}
```

- Here when calling the method displayMessage, we supply a value for the parameter courseName. We call such values **arguments**.
- **(Parameter vs. Argument，形式参数 vs. 实际参数)** A parameter is the variable which is part of the method's declaration. An argument is an expression used when calling the method.

More on Instance Variables

- An object has attributes (e.g., the amount of gas of a car) that are carried with the object as it is used in a program.
- Such attributes exist before a method is called on an object and after the method completes execution.
- A class typically consists of one or more methods that manipulate the attributes that belong to a particular object of the class.
- Attributes are represented as variables in a class declaration.

More on Instance Variables

Object attributes are represented as variables (called **fields**) in a class declaration.

```
public class GradeBook {
   private String courseName;
   public void displayMessage( String courseName ) {
      System.out.println("Welcome to the Grade Book for
      the course%s!\n“, courseName);
   }
}
```

More on Instance Variables

Each object (instance) of the class has its own copy of an attribute in memory, the **field** that represents the attribute is also know as an **instance variable**.

```
public class GradeBook {
   private String courseName;
   public void displayMessage( String courseName ) {
      System.out.println("Welcome to the Grade Book for
      the course%s!\n“, courseName);
   }
}
```

Don’t Confuse with Local Variables

```
public class GradeBookTest {
    public static void main(String[] args) {
        GradeBook myGradeBook = new GradeBook();
        myGradeBook.displayMessage(“Java Programming”);
    }
}
```

Variables declared in the body of a particular method are known as **local variables** and can be only used in that method.

Instance variables are declared inside a class declaration, but outside the bodies of the class’ method declarations.

Declaring Methods to Manipulate Instance Variables

```
public class GradeBook {
   private String courseName;

   // method to set the course name
   public void setCourseName(String name) {
      courseName = name;
   }

   // method to retrieve the course name
   public String getCourseName() {
      return courseName;
   }
}
```

Access Modifiers: Public & Private

Most instance variables are declared to be **private** (data hiding). Variables (or methods) declared to be private are accessible only to methods of the class in which they are declared.

```
public class GradeBook {
   private String courseName;

   public void setCourseName(String name) {
      courseName = name;
   }

   public String getCourseName() {
      return courseName;
   }
}
```

Using Getter and Setter

```
public class GradeBook {
   private String courseName;

   public void setCourseName(String name) {
      courseName = name;
   }

   public String getCourseName() {
      return courseName;
   }
   public void displayMessage() {
      System.out.printf(“Welcome to the grade book
      for\n%s!\n”, getCourseName());
   }
}
```

Manipulate the value of fields

Retrieve the value of fields

Using Getter and Setter

```
public class GradeBook {
   …
   public void displayMessage() {
      System.out.printf(“Welcome to the grade book
      for\n%s!\n”, getCourseName());
   }
}
```

```
import java.util.Scanner;
public class GradeBookTest {
    public static void main(String[] args) {
        GradeBook myGradeBook = new GradeBook();
        Scanner input = new Scanner(System.in);
        System.out.printf(“Enter course name: ”);
        String theName = input.nextLine();
        myGradeBook.setCourseName(theName);
        System.out.println();
        myGradeBook.displayMessage();
    }
}
```

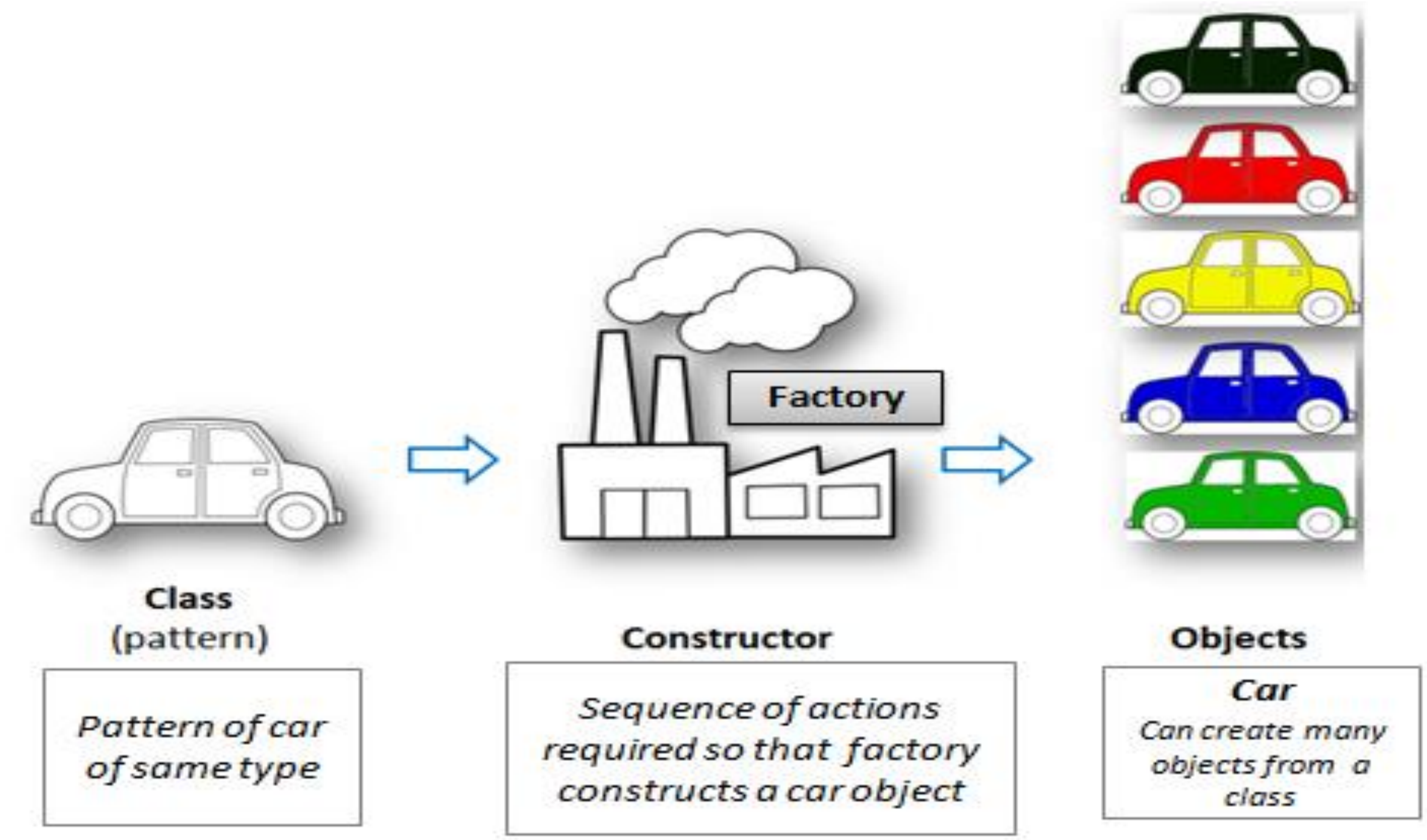
```
Enter course name: CS102A

Welcome to the grade book for
CS102A
```

Initializing Objects with Constructors

- Each class you declare can provide a special method called a constructor (构造方法) that can be used to initialize an object of a class when the object is created

Initializing Objects with Constructors


Factory
Class
(pattern)
Pattern of car of same type
Constructor
Sequence of actions required so that factory constructs a car object
Objects
Car
Can create many objects from a class

Initializing Objects with Constructors

- Each class you declare can provide a special method called a constructor that can be used to initialize an object of a class when the object is created
- Java requires a constructor call for *every* object that is created
- Keyword `new` requests memory from the system to store an object, then calls the corresponding class's constructor to initialize the object.

```
GradeBook myGradeBook = new GradeBook();
```

Initializing Objects with Constructors

```
GradeBook myGradeBook = new GradeBook();
```

- The empty parentheses after "`new GradeBook`" indicate a call to the class's constructor without arguments
- The compiler provides a default constructor with no parameters in any class that does not explicitly include a constructor
  - When a class has only the default constructor, its instance variables are initialized with default values (e.g., an `int` variable gets the value 0)
- When you declare a class, you can provide your own constructor to specify custom initialization for objects of your class

```
public class GradeBook {

   private String courseName;  // course name for this Gradebook

   public GradeBook( String name ){
      courseName = name;  // initialize courseName
   } // end constructor

   // method to set the course name
   public void setCourseName( String name ) {
      courseName = name;  // store the course name
   }  // end method setCourseName
}
```

Initializing Objects with Constructors

```
public GradeBook(String name) {
     courseName = name; // initialize courseName
} // end constructor
```

- Like a method, a constructor's parameter list specifies the data it requires to perform its task.
    - When creating a new object, the data is placed in the parentheses after the class name: `GradeBook book = new GradeBook("CS102A");`
- A class instance creation expression returns a **reference** to the new object (the address to its variables and methods in memory).

```
public class GradeBookTest {
// main method for the program
    public static void main( String[] args ) {
    // create GradeBook object
    GradeBook gradebook1 = new GradeBook(
    "CS101 Introduction to Java Programming");
    GradeBook gradebook2 = new GradeBook(
    "CS102 Object-oriented Programming in Java");

    // display initial value of CourseName for each GradeBook
        System.out.printf("gradebook1 course name is %s.\n",
gradebook1.getCourseName());
        System.out.printf("gradebook2 course name is %s.\n",
gradebook2.getCourseName());
    }  // end main
} // end class GradeBookTest
```

Initializing Objects with Constructors

- An important difference between constructors and methods is that constructors cannot return values, so they cannot specify a return type (not even `void`).
- Normally, constructors are declared `public`.
- *If you declare any constructors for a class, the Java compiler will not create a default constructor for the class.*

More on Default Constructors

```
public class GradeBook { // no constructor provided by the programmer
   private String courseName;
   public void setCourseName(String name) {
      courseName = name;
   }
   public String getCourseName() {
      return courseName;
   }
   public void displayMessage() {
      System.out.printf("Welcome to the grade book for\n%s!\n", getCourseName());
   }
}
```

Can we write the following statement to create a GradeBook object?

```
GradeBook myGradeBook = new GradeBook();
```

Yes. Compiler will provide a default constructor with no parameters.

More on Default Constructors

```
public class GradeBook { // this version has a constructor
   private String courseName;
   public GradeBook(String name) {
      courseName = name;
   }
   public void setCourseName(String name) {
      courseName = name;
   }
   public String getCourseName() {
      return courseName;
   } …
}
```

Can we write the following statement to create a GradeBook object?

```
GradeBook myGradeBook = new GradeBook();
```

No. Compiler will not provide a default constructor this time. The statement will cause a **compilation error**.

Case Study: Account Balances

- We define a class named `Account` to maintain the balance of a bank account.

```
public class Account {

    private String clientName;  // instance variable that stores
client name for this account
    private double balance;  // instance variable that stores the
balance

    // Account
    public Account(String name, double initialBalance){
        clientName = name;  // initial clientName
        if(initialBalance >= 0.0){
            balance = initialBalance;  // initialize balance
        }
    } // end constructor

    // method to deposit an amount to the account
    public void deposit( double amount ) {
        balance += amount;  // add amount to the balance
    }  // end method deposit
    // return the account balance
    public double getBalance() {
      return balance;
    }  // end method getBalance

    // return the account balance
    public String getName() {
      return clientName;
    }  // end method getBalance

    public void displayAccount() {
        System.out.printf("This account belongs to %s, with a balance
of %f.\n", getName(), getBalance());
    }
}
```

Validating Constructor Arguments

- It's common for someone to open an account to deposit money immediately, so the constructor receives a parameter `initialBalance` of type `double` that represents the initial balance.
  - Validate the constructor argument to ensure that `initialBalance` is greater than `0.0`
  - If so, `initialBalance`'s value is assigned to instance variable `balance.`
  - Otherwise, `balance` remains at `0.0` (its default initial value).

```
// Account
    public Account(String name, double initialBalance){
        clientName = name;  // initial clientName
        if(initialBalance >= 0.0){
            balance = initialBalance;  // initialize balance
        }
    }// end constructor
```

Case Study: Account Balances; Validating Constructor Arguments

- We further define a class `AccountTest` that creates and manipulates two `Account` objects.

```
import java.util.Scanner;

public class AccountTest {
// main method for the program
   public static void main( String[] args ) {
      // create GradeBook object
      Account account1 = new Account("San Zhang", 10.0);
      Account account2 = new Account("Si Li", 50.3);

      // display balances
      account1.displayAccount();
      account2.displayAccount();

      // create a Scanner to obtain input from the command window
      Scanner input = new Scanner(System.in);
      System.out.print("Enter deposit amount for account1: "); // prompt
      double depositAmount = input.nextDouble(); // obtain user input
      System.out.printf("\nadding %.2f to account1 balance\n\n",
depositAmount);
      account1.deposit(depositAmount); // add to account1's balance
      // display balances
      account1.displayAccount();
      account2.displayAccount();

      System.out.print("Enter deposit amount for account2: "); // prompt
      depositAmount = input.nextDouble(); // obtain user input
      System.out.printf("\nadding %.2f to account2 balance\n\n", 
depositAmount);
      account2.deposit(depositAmount); // add to account1's balance

      // display balances
      account1.displayAccount();
      account2.displayAccount();
   }  // end main
} // end class GradeBookTest
```

```
This account belongs to San Zhang, with a balance of 10.000000.
This account belongs to Si Li, with a balance of 50.300000.
Enter deposit amount for account1: 50

adding 50.00 to account1 balance

This account belongs to San Zhang, with a balance of 60.000000.
This account belongs to Si Li, with a balance of 50.300000.
Enter deposit amount for account2: 30

adding 30.00 to account2 balance

This account belongs to San Zhang, with a balance of 60.000000.
This account belongs to Si Li, with a balance of 80.300000.
```

Java API Packages

- Application Programming Interface (API)
- Java contains many predefined classes that are grouped into categories of related classes called packages
- Overview of the packages in Java SE 8
  - https://docs.oracle.com/javase/8/docs/api/

Commonly-used packages

`java.io`	The Java Input/Output Package contains classes and interfaces that enable programs to input and output data. (See Chapter 17, Files, Streams and Object Serialization.)
`java.lang`	The Java Language Package contains classes and interfaces (discussed bookwide) that are required by many Java programs. This package is imported by the compiler into all programs.
`java.net`	The Java Networking Package contains classes and interfaces that enable programs to communicate via computer networks like the Internet. (See Chapter 27, Networking.)
`java.sql`	The JDBC Package contains classes and interfaces for working with databases. (See Chapter 28, Accessing Databases with JDBC.)
`java.util`	The Java Utilities Package contains utility classes and interfaces that enable such actions as date and time manipulations, random-number processing (class `Random`, Section 5.9) and the storing and processing of large amounts of data.
`java.util.concurrent`	The Java Concurrency Package contains utility classes and interfaces for implementing programs that can perform multiple tasks in parallel. (See Chapter 26, Multithreading.)

Assignment 2

- Due on 10/31/2021, 22:00
- Submit on OJ