Vietnam National University of HCMC
International University
School of Computer Science and Engineering

Classes & Objects in Java (IT069IU)

Nguyen Trung Ky

ntky@hcmiu.edu.vn

it.hcmiu.edu.vn/user/ntky

Previously,

We talked about:

- Different Java Platforms
- Our choice of JDK
- The best IDE selections
- Create the first Java programs
- Compile and Run with commands or on Eclipse
- Java data types:
  - Primitive
  - Non-primitive (reference types)
- Variable
- Operators

Agenda

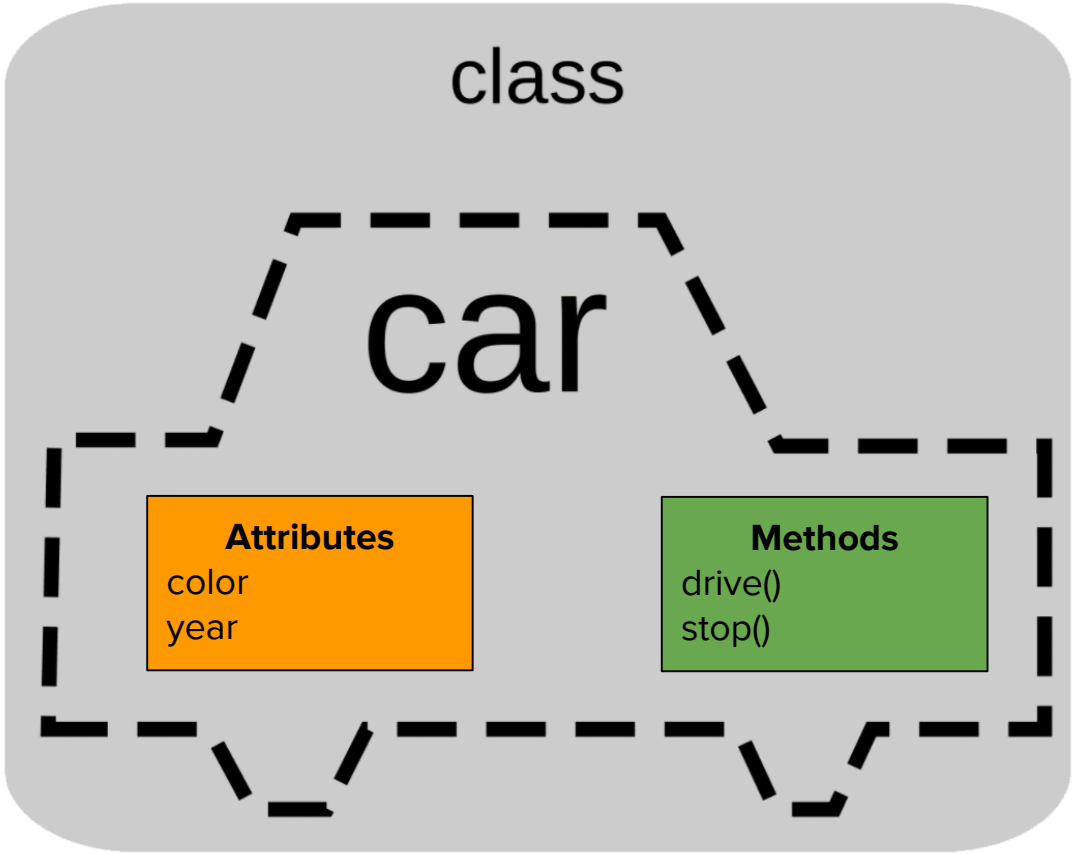
- **Class**
  - **Attributes**
  - **Method** with Parameters
    - **Getters** and **Setters** Methods
  - **Access Modifiers** (Public & Private)
  - **Constructor** with Parameters
  - **UML** Diagram
- **Object**
  - **Create objects** from class with keyword **new**
  - Call **methods** with input argument
- **Primitive** vs **Reference**
- **Useful Classes**
  - **Scanner**
    - Read input string with nextLine(), next()
    - Read input number with nextDouble()
  - **String**
    - Display string with print(), println(), printf()
  - **Math**
    - pow(), max(), random()
- Bank account application example

Class & Object

Class

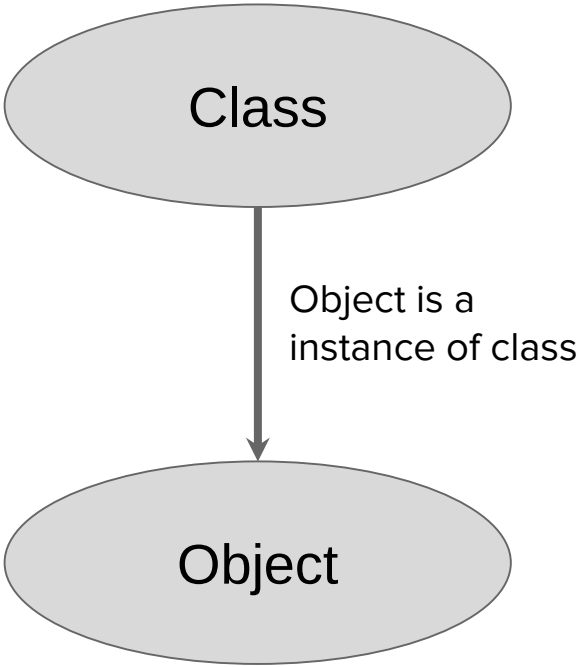
- Class is a **blueprint** for an object.
- Class is used to **create objects**.
- Class **define** what kind of **attributes and methods** for all the objects of the same class to have.
- Class can be considered to be a **factory** which **produce** a specific type of **objects**.

Class Car

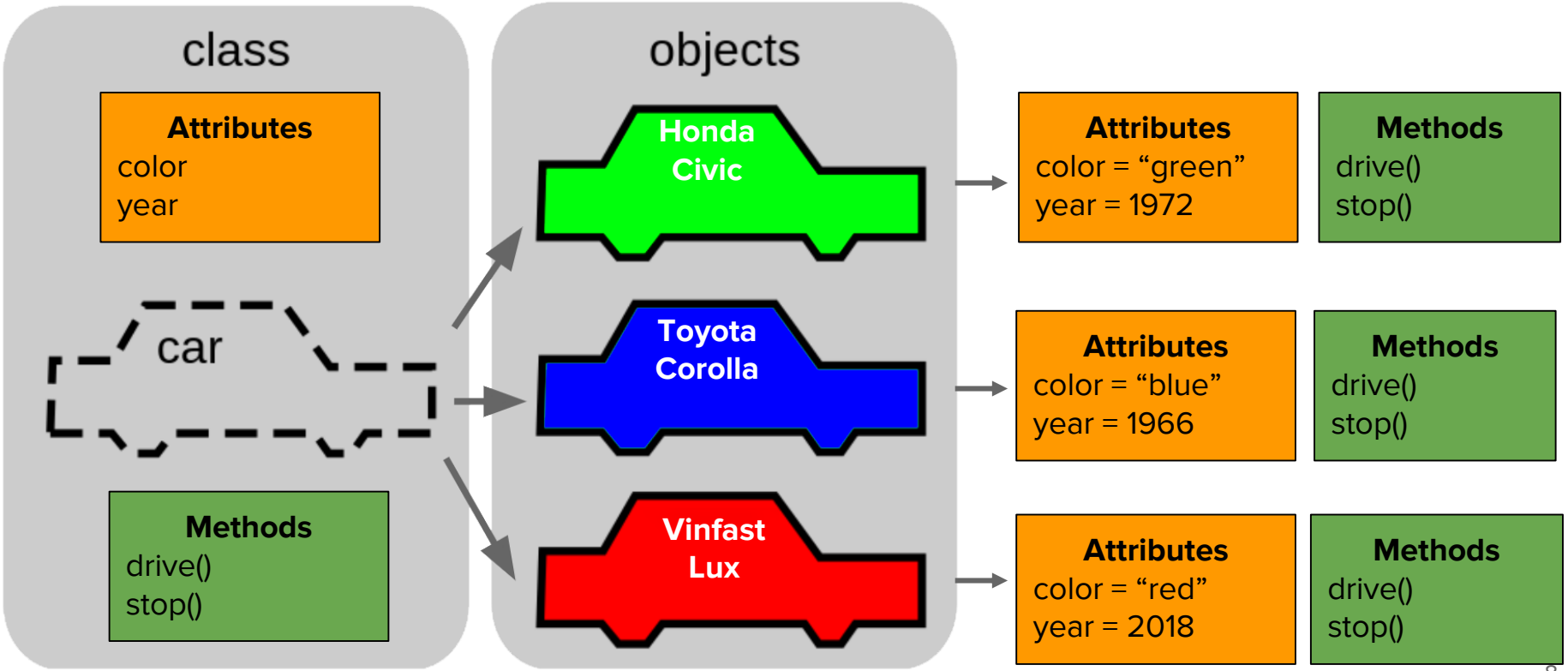


Object

- Objects can be **created** from a class.
- Object is the **instance** of that class if that object is created by that class.
- Each Object has a different **state** (different values for their attributes).
- Objects can **send messages** to each other by calling (**invoke**) **methods** of each others.



Objects from Car Class



Class Declration in Java

Syntax 1: use keyword class

```
class <Class_name>
{
        // content
}
```

Car.java

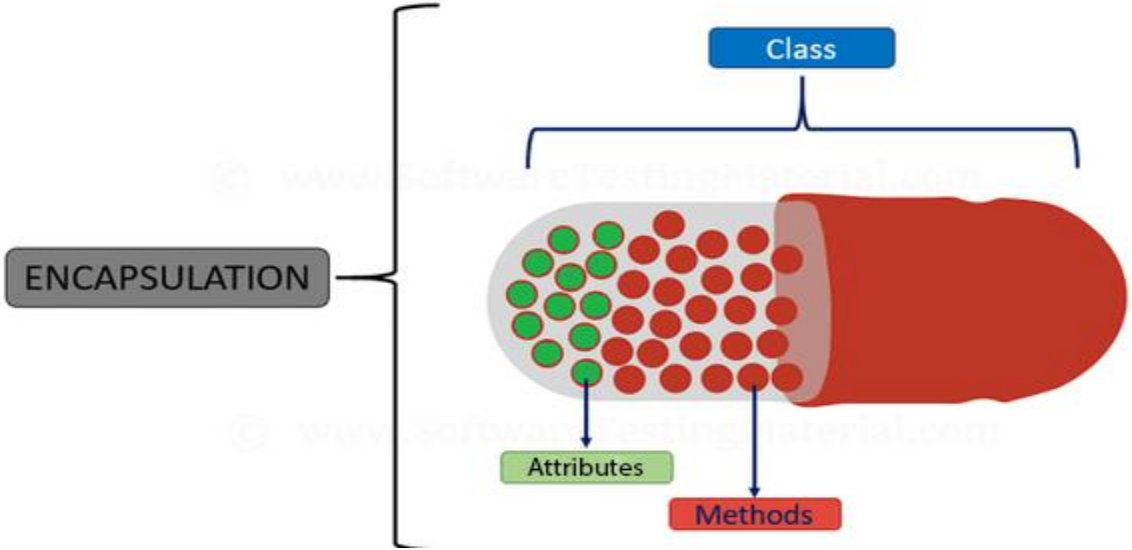
```java
class Car {
    String color;
    int year;

    Car(String color, int year) {
        this.color = color;
        this.year = year;
    }

    void drive(){
        System.out.println("Car is driving now!");
    }

    void stop(){
        System.out.println("Car stopped!");
    }
}
```

CarFactory.java

```java
class CarFactory {
    public static void main(String[] args) {
        Car hondaCivic = new Car( color: "green",  year: 1972);
        Car toyotaCorolla = new Car( color: "blue",  year: 1966);
        Car vinfastLux = new Car( color: "red",  year: 2018);

        hondaCivic.drive();
        toyotaCorolla.drive();
        vinfastLux.drive();

        hondaCivic.stop();
        toyotaCorolla.stop();
        vinfastLux.stop();
    }
}
```

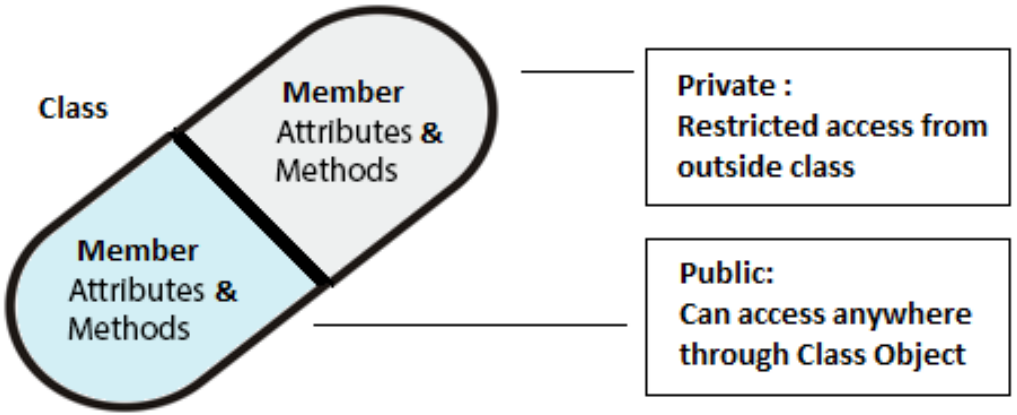
Encapsulation

- Encapsulation is a process of **wrapping all relevant attributes and methods** into a **single class**, and define the level access of the classes through **access modifier**.
- A class is **fully encapsulated** when all the data members (attributes) are **private**.



Access Modifiers

- Attributes (data) and Methods (behavior) are **members** of an object.
- **The access** to members of an object can be **controlled (Data Hiding**)
- **Public**: Those members can be **anywhere.**
- **Private**: Those members can be **only within the class**.
- No restrictions on member access is a bad OO designed.



Access Modifier

Modifier	Class	Package	Subclass	Global
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
Private	✓	✗	✗	✗

Class Declration in Java

Syntax 2: use access modifier

```
access modifier class <Class_name>
{
        // Body
}
```

Example of GradeBook Class

GradeBook.java

```
1  // Fig. 3.1: GradeBook.java
2  // Class declaration with one method.
3
4  public class GradeBook
5  {
6     // display a welcome message to the GradeBook user
7     public void displayMessage()
8     {
9        System.out.println( "Welcome to the Grade Book!" );
10    } // end method displayMessage
11 } // end class GradeBook
```

Performs the task of displaying a message on the screen; method `displayMessage` must be called to perform this task

Fig. 3.1 | Class declaration with one method.

Explanation for GradeBook Class

- Create a new class, **GradeBook**, as a template to create objects of that type latter.
- Each class declaration that begins with keyword **public** must be stored in a file that has the **same name** as the class and ends with the **.java** file-name extension.
- Keyword **public** is an access modifier.
  - Indicates that the class is “available to the public”
- The **return type** specifies the type of data the method returns after performing its task.
- Return type **void** indicates that a method will perform a task but **will not return** (i.e., give back) any information to its calling method when it completes its task.
- Cannot execute Class GradeBook because it **does not contain main method**.
  - Otherwise will receive an error message like:
    - Exception in thread "main" java.lang.NoSuchMethodError: main

GradeBookTest Class with main method

GradeBookTest.java

```
1  // Fig. 3.2: GradeBookTest.java
2  // Creating a GradeBook object and calling its displayMessage method.
3
4  public class GradeBookTest
5  {
6     // main method begins program execution
7     public static void main( String[] args )
8     {
9        // create a GradeBook object and assign it to myGradeBook
10       GradeBook myGradeBook = new GradeBook();
11
12       // call myGradeBook's displayMessage method
13       myGradeBook.displayMessage();
14    } // end main
15 } // end class GradeBookTest
```

Creates a `GradeBook` object and assigns it to variable `myGradeBook`

Invokes method `displayMessage` on the GradeBook object that was assigned to variable `myGradeBook`

```
Welcome to the Grade Book!
```

Fig. 3.2 | Creating a `GradeBook` object and calling its `displayMessage` method.

Explanation for GradeBookTest Class

- **Public static void main(String args[])** is the **entry (starting) point** of the whole project which is **compulsory**.
- To help you prepare for the larger programs, use a separate class (GradeBookTest) containing method **main** to **test each new class**.
  - Some programmers refer to such a class as a **driver class**.
- The **main method** is **called automatically** by the Java Virtual Machine (**JVM**) when you execute an application.

Class Objects Creation

- **Class objects can be created from a class**:
  - Keyword **new** creates a new object of the class.
  - The parentheses to the right of the class name are required.
  - Parentheses in combination with a class name represent **a call to a constructor**, which is similar to a method but is used only at the time an object is created to initialize the object's data.
  - In other words, **it instantiates a class by allocating memory for a new object and returning a reference to that memory**.

```
// create a GradeBook object and assign it to myGradeBook
GradeBook myGradeBook = new GradeBook();
```

Creates a GradeBook object and assigns it to variable myGradeBook

Call Methods of an Class Object

- Call a method via the class-type variable
  - Variable name followed by a dot separator (.), the method name and parentheses.
  - Call the method to perform its task.
- Any class can contain a main method.
  - The JVM invokes the main method only in the class used to execute the application.
  - If multiple classes that contain main, then one that is invoked is the one in the class named in the java command.

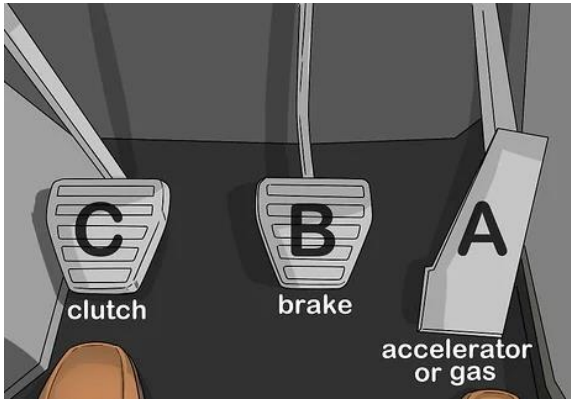
```
// call myGradeBook's displayMessage method
myGradeBook.displayMessage();
```

Invokes method `displayMessage` on the GradeBook object that was assigned to variable `myGradeBook`

Method Parameter

- Car analogy
  - Pressing a car's gas pedal sends a message to the car to perform a task—make the car go faster.
  - The farther down you press the pedal, the faster the car accelerates.
  - Message to the car includes the task to perform and additional information that helps the car perform the task.
- Parameter: Additional information which a method needs to perform its task.

- A method can require **none, or one, or more parameters.**
- A method call provide values for those parameters, we call them **arguments**.



A Method with a Parameter

GradeBook.java

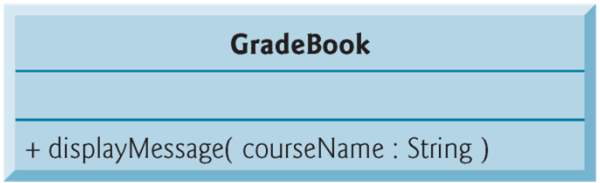
```
1  // Fig. 3.4: GradeBook.java
2  // Class declaration with a method that has a parameter.
3
4  public class GradeBook
5  {
6     // display a welcome message to the GradeBook user
7     public void displayMessage( String courseName )
8     {
9        System.out.printf( "Welcome to the grade book for\n%s!\n",
10          courseName );
11    }
12 } // end class GradeBook
```

Parameter `courseName` provides the additional information that the method requires to perform its task

Parameter `courseName`'s value is displayed as part of the output

Fig. 3.4 | Class declaration with one method that has a parameter.

UML Diagram for GradeBook Class



[Question] Have you noticed any difference between this Gradebook class UML with the previous one?

GradeBookTest.java

```
1  // Fig. 3.5: GradeBookTest.java
2  // Create GradeBook object and pass a String to
3  // its displayMessage method.
4  import java.util.Scanner; // program uses Scanner
5
6  public class GradeBookTest
7  {
8     // main method begins program execution
9     public static void main( String[] args )
10    {
11       // create Scanner to obtain input from command window
12       Scanner input = new Scanner( System.in );
13
14       // create a GradeBook object and assign it to myGradeBook
15       GradeBook myGradeBook = new GradeBook();
16
17       // prompt for and input course name
18       System.out.println( "Please enter the course name:" );
19       String nameOfCourse = input.nextLine(); // read a line of text
20       System.out.println(); // outputs a blank line
21
22       // call myGradeBook's displayMessage method
23       // and pass nameOfCourse as an argument
24       myGradeBook.displayMessage( nameOfCourse );
25    } // end main
26 } // end class GradeBookTest
```

Reads a `String` from the user

Passes the value of `nameOfCourse` as the argument to method `displayMessage`

Output:

```
Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

Instance Variables

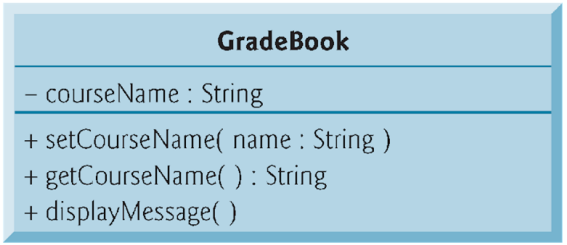
- A class normally consists of one or more **methods** that **manipulate the attributes** that belong to a particular object of the class.
  - **Attributes** (**fields**) are represented as **variables** in a class declaration.
  - Declared inside a class body but outside the bodies of methods.
- **Instance variable**
  - When each object of a class maintains its own copy of an attribute (field/instance variable).

```
public class GradeBook
{
   private String courseName; // course name for this GradeBook
```

Each GradeBook object maintains its own copy of instance variable courseName

Private Attributes, Getters & Setters Methods

- Instance variables (attributes) should be **private** (**data hiding** or **information hiding**) and **methods** to be are **public**.
  - **Private variables** are **encapsulated (hidden)** in the object and only are **accessible only** to methods of the class.
    - Prevents instance variables from being modified accidentally by a class in another part of the program.
    - Set and get methods:
      - **Setters**: methods to **change the values** of (private) variable
      - **Getters**: method used to **access the values** of (private) variables.
    - Sometimes, methods can be private only they are only needed to be accessed by other methods of the same class. (helper methods)



Attributes (Fields) in GradeBook Class

GradeBook.java

```
1  // Fig. 3.7: GradeBook.java
2  // GradeBook class that contains a courseName instance variable
3  // and methods to set and get its value.
4
5  public class GradeBook
6  {
7     private String courseName; // course name for this GradeBook
8
9     // method to set the course name
10    public void setCourseName( String name )
11    {
12       courseName = name; // store the course name
13    } // end method setCourseName
14
15    // method to retrieve the course name
16    public String getCourseName()
17    {
18       return courseName;
19    } // end method getCourseName
20
21    // display a welcome message to the GradeBook user
22    public void displayMessage()
23    {
24       // calls getCourseName to get the name of
25       // the course this GradeBook represents
26       System.out.printf( "Welcome to the grade book for\n%s!\n",
27          getCourseName() );
28    } // end method displayMessage
29 } // end class GradeBook
```

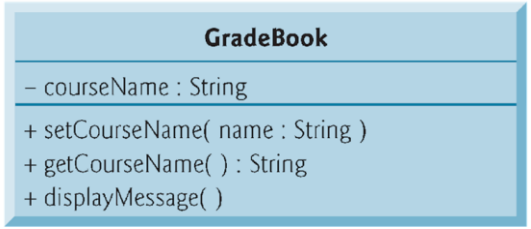
Each `GradeBook` object maintains its own copy of instance variable `courseName`

Method allows client code to change the `courseName`

Method allows client code to obtain the `courseName`

No parameter required; all methods in this class already know about instance variable `courseName` and the class's other methods

Good practice to access your instance variables via *set* or *get* methods



GradeBookTest.java

```
1  // Fig. 3.8: GradeBookTest.java
2  // Creating and manipulating a GradeBook object.
3  import java.util.Scanner; // program uses Scanner
4
5  public class GradeBookTest
6  {
7     // main method begins program execution
8     public static void main( String[] args )
9     {
10       // create Scanner to obtain input from command window
11       Scanner input = new Scanner( System.in );
12
13       // create a GradeBook object and assign it to myGradeBook
14       GradeBook myGradeBook = new GradeBook();
15
16       // display initial value of courseName
17       System.out.printf( "Initial course name is: %s\n\n",
18          myGradeBook.getCourseName() );
19
20       // prompt for and read course name
21       System.out.println( "Please enter the course name:" );
22       String theName = input.nextLine(); // read a line of text
23       myGradeBook.setCourseName( theName ); // set the course name
24       System.out.println(); // outputs a blank line
25
26       // display welcome message after specifying course name
27       myGradeBook.displayMessage();
28    } // end main
29 } // end class GradeBookTest
```

Gets the value of the myGradeBook object's courseName instance variable

Sets the value of the courseName instance variable

Displays the GradeBook's message, including the value of the courseName instance variable

Output:

```
Initial course name is: null

Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

[Question]

- Why is the course name is null at the beginning?
- Is there any way to set variable value of an object right away when creating it?

Constructors in a Class

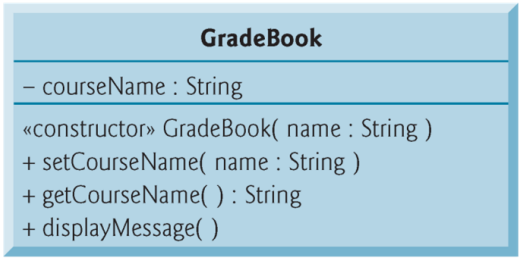
- A **constructor** is **called** when we use the **keyword new** to create a new object.
  - Keyword new requests memory from the system to store an object, then calls the corresponding class's constructor to initialize the object.
- Java **requires a constructor call for every object that is created**.
  - **Without providing your constructor**, a class will have a **default constructor** with no parameters**.**
    - All variables will be initialized to be **default values** when that object is created.
  - **With your own constructors**:
    - A constructor of a class can set variable values as input arguments when the object is created.
- A **constructor** is a special **public method**, which must have the **same name** as the **class**.

```
public class GradeBook
{
   private String courseName; // course name for this GradeBook

   // constructor initializes courseName with String argument
   public GradeBook( String name )
   {
      courseName = name; // initializes courseName
   } // end constructor
```

Constructor that initializes courseName to the specified value

Constructor in a Class

GradeBook.java




```
1  // Fig. 3.10: GradeBook.java
2  // GradeBook class with a constructor to initialize the course name.
3
4  public class GradeBook
5  {
6     private String courseName; // course name for this GradeBook
7
8     // constructor initializes courseName with String argument
9     public GradeBook( String name )
10    {
11       courseName = name; // initializes courseName
12    } // end constructor
13
14    // method to set the course name
15    public void setCourseName( String name )
16    {
17       courseName = name; // store the course name
18    } // end method setCourseName
19
```

Constructor that initializes `courseName` to the specified value

Fig. 3.10 | `GradeBook` class with a constructor to initialize the course name. (Part 1 of 2.)

GradeBook.java

```
20      // method to retrieve the course name
21      public String getCourseName()
22      {
23         return courseName;
24      } // end method getCourseName
25
26      // display a welcome message to the GradeBook user
27      public void displayMessage()
28      {
29         // this statement calls getCourseName to get the
30         // name of the course this GradeBook represents
31         System.out.printf( "Welcome to the grade book for\n%s!\n",
32            getCourseName() );
33      } // end method displayMessage
34   } // end class GradeBook
```

Fig. 3.10 | GradeBook class with a constructor to initialize the course name. (Part 2 of 2.)

GradeBookTest.java

```
1  // Fig. 3.11: GradeBookTest.java
2  // GradeBook constructor used to specify the course name at the
3  // time each GradeBook object is created.
4
5  public class GradeBookTest
6  {
7     // main method begins program execution
8     public static void main( String[] args )
9     {
10       // create GradeBook object
11       GradeBook gradeBook1 = new GradeBook(
12          "CS101 Introduction to Java Programming" );
13       GradeBook gradeBook2 = new GradeBook(
14          "CS102 Data Structures in Java" );
15
16       // display initial value of courseName for each GradeBook
17       System.out.printf( "gradeBook1 course name is: %s\n",
18          gradeBook1.getCourseName() );
19       System.out.printf( "gradeBook2 course name is: %s\n",
20          gradeBook2.getCourseName() );
21    } // end main
22 } // end class GradeBookTest
```

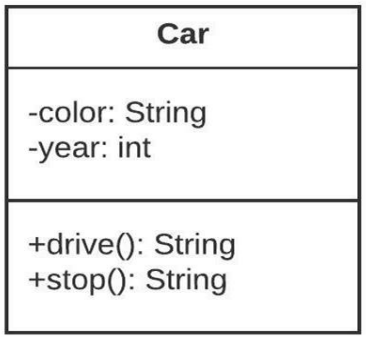
Class instance creation expression initializes the `GradeBook` and returns a reference that is assigned to variable `gradeBook1`

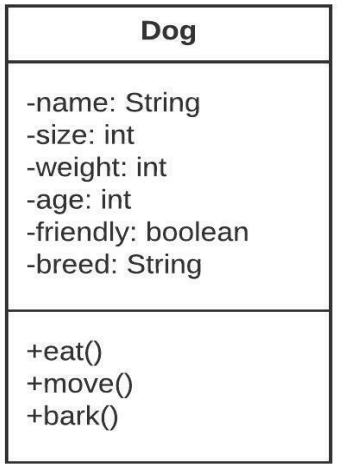
Class instance creation expression initializes the `GradeBook` and returns a reference that is assigned to variable `gradeBook1`

```
gradeBook1 course name is: CS101 Introduction to Java Programming
gradeBook2 course name is: CS102 Data Structures in Java
```

UML (Unified Modeling Language)

- A diagram where people describe the design of the projects by showing the classes, their attributes, methods, and the relationships among those objects.
- Each class is modeled in a class diagram as a rectangle with three compartments.
  - **Top**: contains the class name centered horizontally in boldface type.
  - **Middle**: contains the class's attributes, which correspond to instance variables.
  - **Bottom**: contains the class's operations, which correspond to methods.
- Access Modifiers: (+) means public and (-) means private.





Class Name

Attributes

Methods

UML class diagram for GradeBook

GradeBook.java

```
1  // Fig. 3.1: GradeBook.java
2  // Class declaration with one method.
3
4  public class GradeBook
5  {
6     // display a welcome message to the GradeBook user
7     public void displayMessage()
8     {
9        System.out.println( "Welcome to the Grade Book!" );
10    } // end method displayMessage
11 } // end class GradeBook
```

Performs the task of displaying a message on the screen; method `displayMessage` must be called to perform this task

Fig. 3.1 | Class declaration with one method.

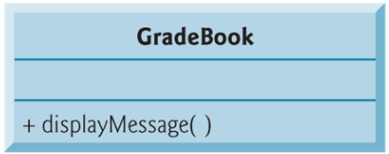
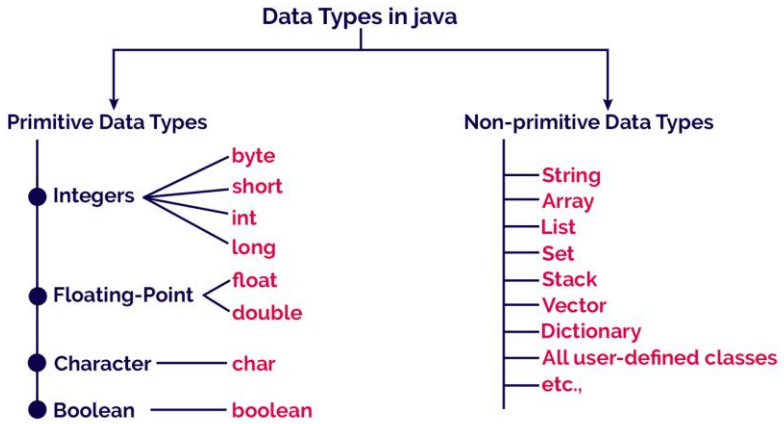


Fig. 3.3 | UML class diagram indicating that class `GradeBook` has a `public` `displayMessage` operation.

Primitives vs References

Primitive types are basic Java types

- int, long, double, boolean, char, short, byte, float.
- A variable stores the actual value of that type.

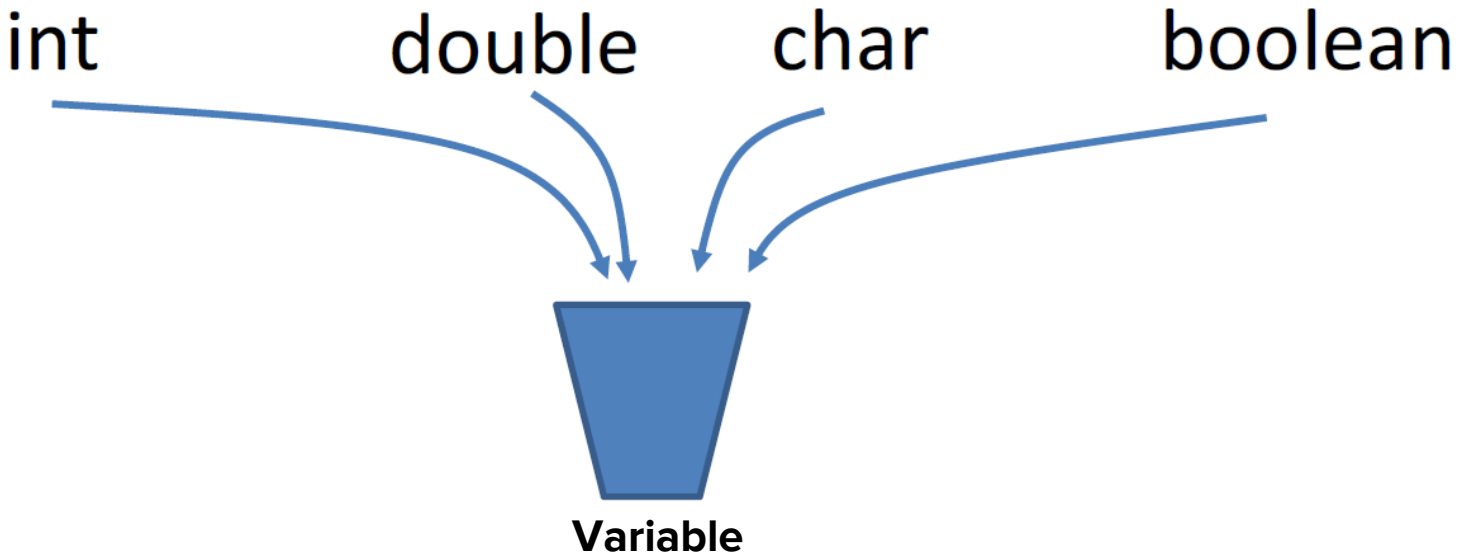


Non-primitive types are **Reference** types

- String, Array, Class Objects, …
- A variable stores only the reference (memory address location) to point to the actual object.

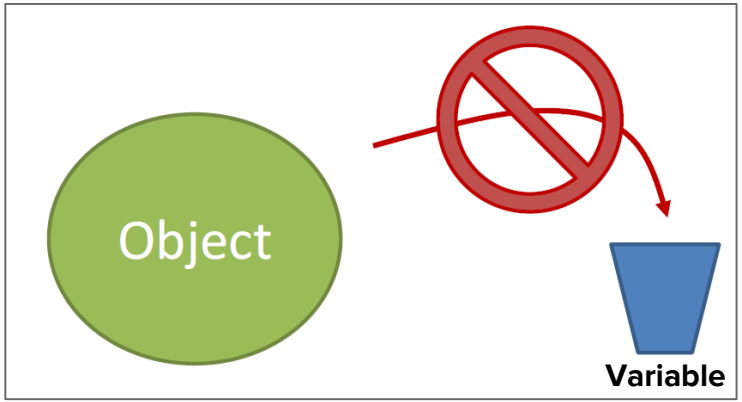
How java stores primitive values

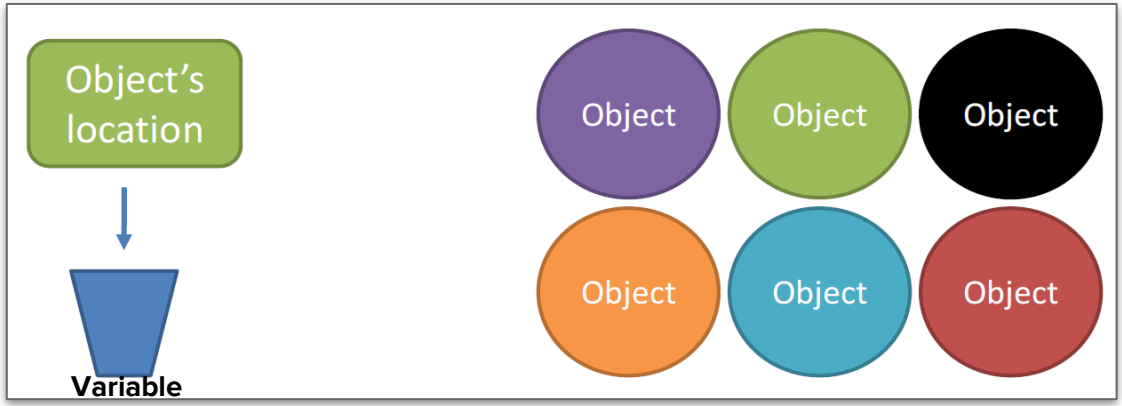
- Variables are like **fixed size cups.**
- Primitives are small enough that they just fit into the cup.
- In other words, the variable stores directly the value of that data.



How java stores objects

- Objects are too big to fit in a variable
  - The actual object is stored somewhere else in the memory.
  - The variable stores only the reference (memory address) which points to the actual object.





References

- The object's location is called a reference
- == operator only compares the references (memory address)

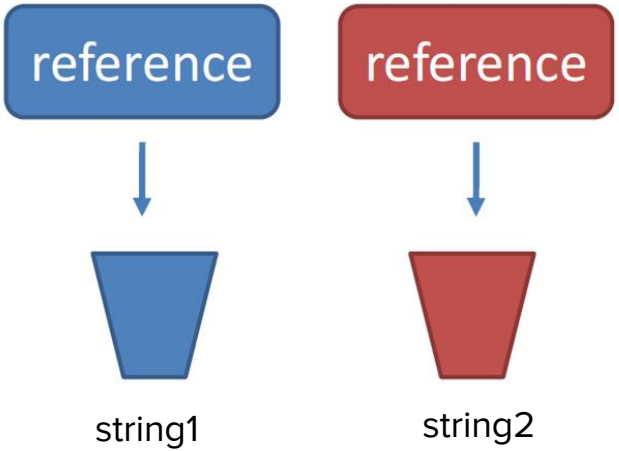
```
String string1 = new String("IU is the best uni");
String string2 = new String("IU is the best uni");

Does string1 == string2?
```

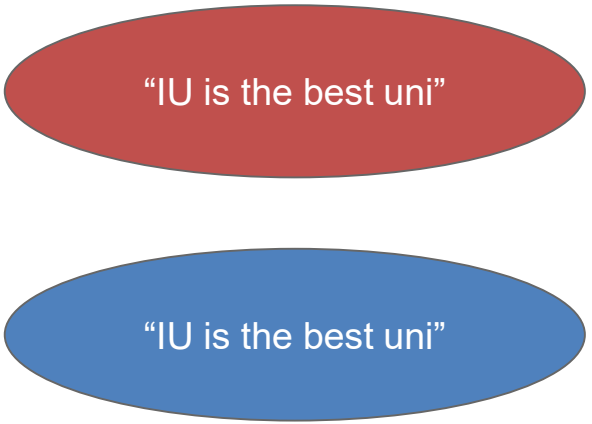
```
Car car1 = new Car("Green", 2020);
Car car2 = new Car("Green", 2020);

Does car1 == car2?
```

References

```
String string1 = new String("IU is the best uni");
String string2 = new String("IU is the best uni");

System.out.print(string1 == string2);
```

[Question] What is the output?

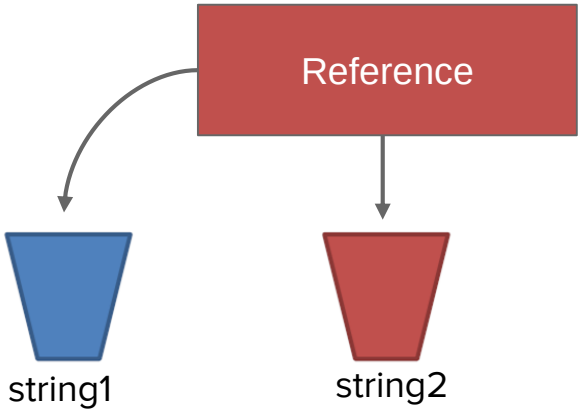


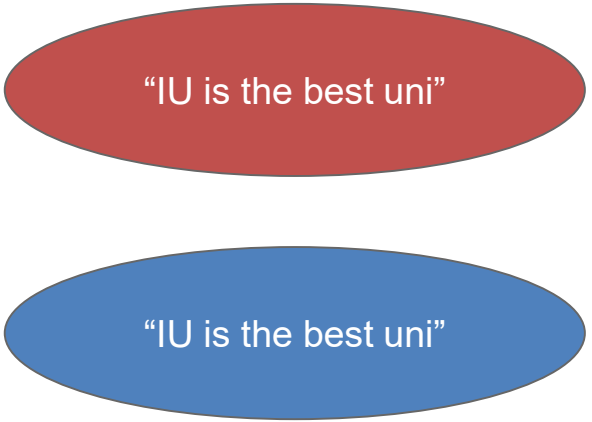


Update References

```
String string1 = new String("IU is the best uni");
String string2 = new String("IU is the best uni");

string1 = string2;    // update the reference

System.out.print(string1 == string2);
```

[Question] What is the output?





Helpful Classes

Reference Types = Objects

Scanner Class: Reading Inputs

- API documentation (http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html)
- To use Scanner class, remember to **import java.util.Scanner**
- Can read from different **input sources**: input keyboards, files, …
- Can convert inputs to different type of **data types**: string, byte, short, int, float, double, …

Java naming convention Method names – lowerCamelCase

String	next(String pattern) Returns the next token if it matches the pattern constructed from the specified string.
BigDecimal	nextBigDecimal() Scans the next token of the input as a BigDecimal.
BigInteger	nextBigInteger() Scans the next token of the input as a BigInteger.
BigInteger	nextBigInteger(int radix) Scans the next token of the input as a BigInteger.
boolean	nextBoolean() Scans the next token of the input into a boolean value and returns that value.
byte	nextByte() Scans the next token of the input as a byte.
byte	nextByte(int radix) Scans the next token of the input as a byte.
double	nextDouble() Scans the next token of the input as a double.
float	nextFloat() Scans the next token of the input as a float.
int	nextInt() Scans the next token of the input as an int.
int	nextInt(int radix) Scans the next token of the input as an int.
String	nextLine()

Scanner Class: Demo

TestScanner.java

```java
import java.util.Scanner;

public class TestScanner {
    public static void main(String[] args) {
        // write your code here
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter name1:");
        String name1 = sc.nextLine();
        System.out.println("name1 entered is '" + name1 +"'.");

        System.out.println("Enter name2: ");
        String name2 = sc.next();
        System.out.println("name 2 entered is '" + name2 +"'.");
    }
}
```

```
Enter name1:IU
name1 entered is 'IU'.
Enter name2:
University
name 2 entered is 'University'.
```

```
Enter name1:Tom Jerry
name1 entered is 'Tom Jerry'.
Enter name2:
Tom Huynh
name 2 entered is 'Tom'.
```

[Question] Can you figure out the difference between **println()** and **print()**? **next()** and **nextLine()**?

Scanner: nextLine() vs next()

- **nextLine()** method:
  - Reads characters typed by the user **until the newline character is encountered**.
  - **Returns a String** containing the characters up to, but **not including the newline**.
  - Press Enter to submit the string to the program. After that, it inserts a newline character at the end of the characters the user typed.
  - The newline character is not included in the input string.
- **next()** method
  - Reads individual words **until a white-space character is encountered**, then **returns a String**, but **not including the white-space**.
  - Information after the first white-space character can be read by other statements that call the Scanner's methods later in the program.

Display Output (System.out)

- **System.out.print** method displays an output string.
- Unlike **println**, **print** does not position the output cursor at the beginning of the next line in the command window.
- The **backslash (\)** is an **escape character**

Escape sequence	Description
\n	Newline character. Example: “My name**\n**is James”. My name is James
\t	Horizontal Tab character. Example: “My name**\t**is James” My name is James
****	Backslash character. Example: “My name **** is James” My name \ is James
\”	Double quote. To print a double quote in a string like: System.out.println(“ **\”**my **\”** example**\”** ”) “my “ example”

Print vs Println Example

```
1  // Fig. 2.3: Welcome2.java
2  // Printing a line of text with multiple statements.
3
4  public class Welcome2
5  {
6     // main method begins execution of Java application
7     public static void main( String[] args )
8     {
9        System.out.print( "Welcome to " );
10       System.out.println( "Java Programming!" );
11    } // end method main
12 } // end class Welcome2
```

Prints `Welcome to` and leaves cursor on same line

Prints `Java Programming!` starting where the cursor was positioned previously, then outputs a newline character

```
Welcome to Java Programming!
```

Fig. 2.3 | Printing a line of text with multiple statements.

Escape Character Example

```
1  // Fig. 2.4: Welcome3.java
2  // Printing multiple lines of text with a single statement.
3
4  public class Welcome3
5  {
6     // main method begins execution of Java application
7     public static void main( String[] args )
8     {
9        System.out.println( "Welcome\nto\nJava\nProgramming!" );
10    } // end method main
11 } // end class Welcome3
```

Each \n moves the output cursor to the next line, where output continues

```
Welcome
to
Java
Programming!
```

Fig. 2.4 | Printing multiple lines of text with a single statement.

Display Output Formatted Text with printf

- **System.out.printf** method.
  - f means “formatted” to displays formatted data

- **Format specifiers** begin with a percent sign **(%)** and are followed by a character that represents the data type.
  - **%s** is a placeholder for **string**.
  - **%d** is a placeholder for **integer**. (byte, short, int, long,..).
  - **%f** is a placeholder for **floating** number (float or double)

Printf Example

```
public class PrintfExample {
  public static void main(String[] args) {
     System.out.printf("First: %s", "Hi there!");
     System.out.printf("Second: %s\n%s.\n", "Welcome to", "IU Uni!");
     System.out.printf("Third: %s %s\t%d\n", "Have fun", "with Java!", 123);
     System.out.printf("Last: %f %.2f", 3.12345, 3.12345);
  }
}
```

Output:

```
First: Hi there!Second: Welcome to
IU Uni!.
Third: Have fun with Java!      123
Last: 3.123450 3.12
```

[Question] What does this print?

```
System.out.printf("%s\n%s\n%s\n", "*", "***",
"*****");
```

String Class: Representation in Text

- API documentation (http://docs.oracle.com/javase/7/docs/api/java/lang/String.html)

charAt()
concat()
equals()
indexOf()
lastIndexOf()
length()
toLowerCase()
toUpperCase()
substring()
trim()

And many more…

int	**indexOf**(int ch) Returns the index within this string of the first occurrence of the specified character.
int	**indexOf**(int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**indexOf**(String str) Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf**(String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
String	**intern**() Returns a canonical representation for the string object.
boolean	**isEmpty**() Returns true if, and only if, **length()** is 0.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(String str) Returns the index within this string of the last occurrence of the specified substring.
int	**lastIndexOf**(String str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
boolean	**matches**(String regex)

String Class: Demo 1

TestString.java

```java
public class TestString {
    public static void main(String[] args) {
        String text = new String( original: "I'm studying IT069IU.");

        System.out.println("text: " + text);
        System.out.println("text.length() = " + text.length());
        System.out.println("text.charAt(5) = " + text.charAt(5));

        System.out.println("text.substring(5,8) = " +
                text.substring(5,8));

        System.out.println("text.indexOf(\"in\") = " +
                text.indexOf("in"));

        String newText = text + "How about you?";
        newText = newText.toUpperCase();
        System.out.println("newText: " + newText);
        if (text.equals(newText))
            System.out.println("text and newText are equal.");
        else
            System.out.println("text and newText are not equal.");
    }
}
```

```
text: I'm studying IT069IU.
text.length() = 21
text.charAt(5) = t
text.substring(5,8) = tud
text.indexOf("in") = 9
newText: I'M STUDYING IT069IU.HOW ABOUT YOU?
text and newText are not equal.
```

[**Question 1**] Can you figure out what is the purpose of:

- length()
- charAt(5)
- substring(5, 8)
- indexOf("in")
- + between two strings
- equals

[**Question 2**] Why do we use .equals() instead of == to compare strings?

String Class: Comparing strings

- As strings are objects, do not use == if you want to check if two strings contain the same text
- Use the equals() method provided in the String class instead.

```
Scanner sc = new Scanner(System.in);
System.out.println("Enter 2 identical strings:");
String str1 = sc.nextLine();
String str2 = sc.nextLine();
System.out.println(str1 == str2);
System.out.println(str1.equals(str2));
System.out.println(str2 == str2);
System.out.println(str2.equals(str2));
```

```
If both of inputs are:
Tom
Tom
```

Question:
Can you guess what is the output of the program?

Math Class: Performing Computation

- API documentation (http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html)
- 2 class attributes:
  - E
  - PI

abs()
ceil()
floor()
hypot()
max()
min()
pow()
random()
sqrt()

And many more…

static double	**abs**(double a) Returns the absolute value of a double value.
static float	**abs**(float a) Returns the absolute value of a float value.
static int	**abs**(int a) Returns the absolute value of an int value.
static long	**abs**(long a) Returns the absolute value of a long value.
static double	**acos**(double a) Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*.
static double	**asin**(double a) Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan**(double a) Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan2**(double y, double x) Returns the angle *theta* from the conversion of rectangular coordinates (x, y) to polar coordinates (r, *theta*).
static double	**cbrt**(double a) Returns the cube root of a double value.
static double	**ceil**(double a) Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer.
static double	**copySign**(double magnitude, double sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static float	**copySign**(float magnitude, float sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static double	**cos**(double a) Returns the trigonometric cosine of an angle.
static double	**cosh**(double x) Returns the hyperbolic cosine of a double value.
static double	**exp**(double a) Returns Euler's number *e* raised to the power of a double value.

Math Class: Demo

TestMath.java

```java
import java.util.Scanner;
public class TestMath {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter 3 values: ");
        double num1 = sc.nextDouble();
        double num2 = sc.nextDouble();
        double num3 = sc.nextDouble();
        System.out.println("Power of two numbers = " + Math.pow(num1, num2));

        double maxNumber= Math.max(Math.max(num1, num2), num3);
        System.out.println("Largest = " + maxNumber);

        System.out.println("Generating a random number: " + Math.random());
        System.out.printf("Number with 2 decimal places: %.2f", maxNumber);
    }
}
```

```
Enter 3 values:
3.2 9.6 5.8
Power of two numbers = 70703.3174808972
Largest = 9.6
Generating a random number: 0.6973061366091079
Number with 2 decimal places: 9.60
```

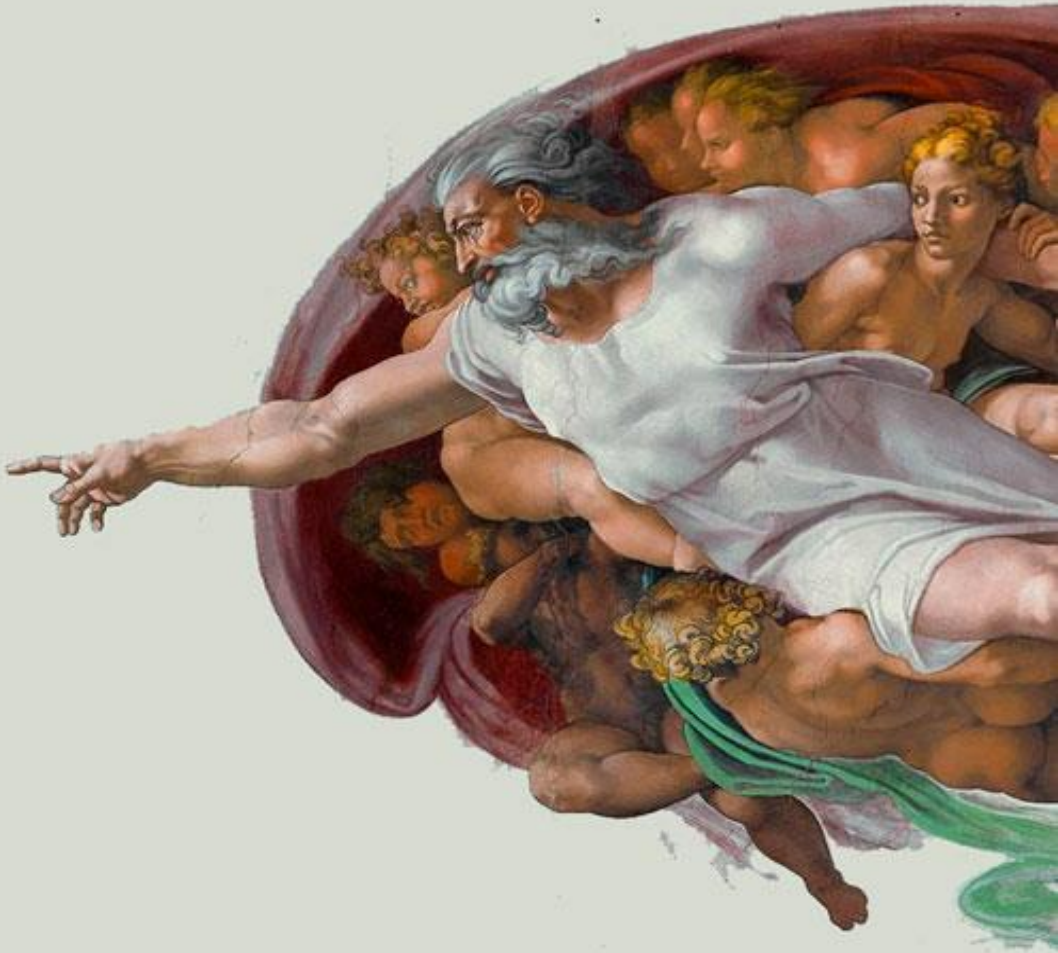
[Question] What is the output range of Math.random() ?

[Question] How to generate a random number between 0 and 10?

[Question] Does Math.random() really generate a true random number?

https://www.baeldung.com/java-generating-random-numbers-in-range

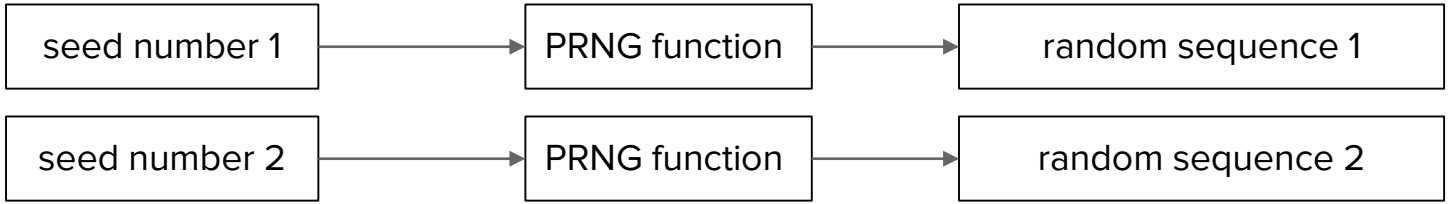
"Tell us what the future holds, so we may know that you are gods."
Isaiah 41:23



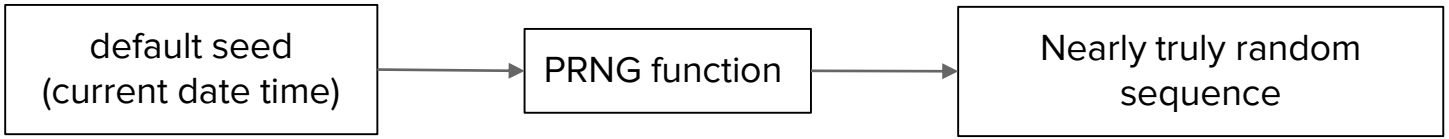
Math.random() is pseudo random number generator (PRNG)

- Using algorithm with an **input seed** to generate a **sequence of numbers** that **approximates randomness**.
- **Pseudo random** because it is **not possible** to generate truly random from **deterministic** thing like computer.

To make the randomness reproducible:



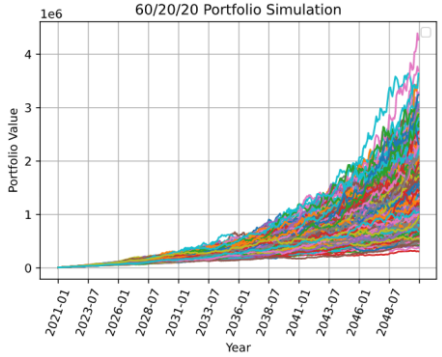
To make the randomness not reproducible:

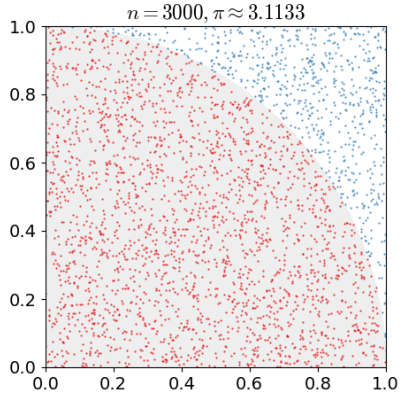


https://www.delftstack.com/howto/java/set-random-seed-in-java/
https://www.journaldev.com/17111/java-random

PRNG Applications

Monte Carlo Simulation





Homework: 649 Lotto Lottery!

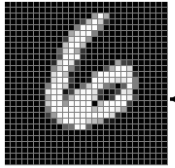
- A lottery **ticket** is **$4**.
- A ticket having **6** different numbers (from **1 to 49**) (Can be repeated)
- On a Saturday, they draw the lottery, and the winning numbers are:

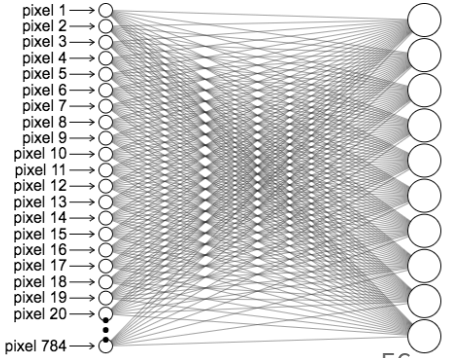
  11, 43, 24, 30, 60, 43

- Match at position:
  - Match **Three** numbers to get a small prize. (**$100**)
  - Matching **Four** numbers gets a bigger prize. (**$1000**)
  - Matching **Five** is even bigger. (**$5000**)
  - Matching **ALL SIX** of the numbers you might win millions. (**$5 million** in cash)
- In the example, we got matches at position 1, 3, 4, 6 (4 numbers) = $1000
- **[Math Question]** What is the possibility of you winning by matching all 6 numbers?
- **Homework Task:**
  - Write a simple program allows you to buy a ticket with six random numbers, and generate the winning numbers and return what kind of prize you won (one game).
  - **Bonus**: imagine you buy up to 100,000 tickets, can you figure out if you actually profit or loss in a long run?

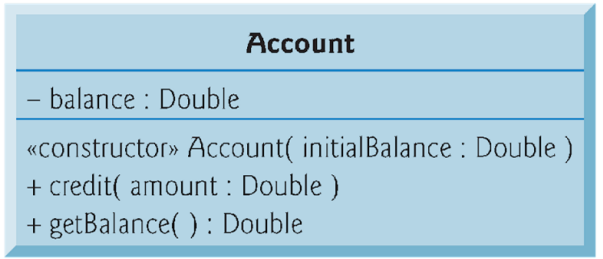
athsisfun.com/data/lottery.html

Machine Learning & Deep Learning





Bank Account Application Example



Write **Two Classes**:

- **Account**:
  - An **attribute** named **"balance"** to store money value (**double** type)
  - A **constructor** to initialize account object with parameter **"initialBalance"** value.
    - Make sure the initial balance number is **only positive number**; **otherwise it is 0.**
  - Two **methods**:
    - **credit**: take an amount as a parameter to add that amount to current balance of that account.
    - **getBalance**: return the current value of the balance of that account.
- **AccountTest**:
  - Create two test Account Objects:
    - **account1** with initial balance to be 50.00
    - **account2** with initial balance to be -7.53
  - **Display the balances** of those two new accounts **after being initialized**.
  - **Take amount values from user inputs** to be added to the balances of the account1 and account2.
  - **Display the balances** of those two accounts before and **after being added new amounts**.

Account.java

```
1  // Fig. 3.13: Account.java
2  // Account class with a constructor to validate and
3  // initialize instance variable balance of type double.
4
5  public class Account
6  {
7     private double balance; // instance variable that stores the balance
8
9     // constructor
10    public Account( double initialBalance )
11    {
12       // validate that initialBalance is greater than 0.0;
13       // if it is not, balance is initialized to the default value 0.0
14       if ( initialBalance > 0.0 )
15          balance = initialBalance;
16    } // end Account constructor
17
18    // credit (add) an amount to the account
19    public void credit( double amount )
20    {
21       balance = balance + amount; // add amount to balance
22    } // end method credit
23
24    // return the account balance
25    public double getBalance()
26    {
27       return balance; // gives the value of balance to the calling method
28    } // end method getBalance
29 } // end class Account
```

Floating-point number for the account balance

Parameter used to initialize the `balance` instance variable

Validating the parameter's value to ensure that it is greater than 0

Initializes `gradeCounter` to 1; indicates first grade about to be input

Returns the value of the `balance` instance variable as a `double`

AccountTest.java

```
1  // Fig. 3.14: AccountTest.java
2  // Inputting and outputting floating-point numbers with Account objects.
3  import java.util.Scanner;
4
5  public class AccountTest
6  {
7     // main method begins execution of Java application
8     public static void main( String[] args )
9     {
10       Account account1 = new Account( 50.00 ); // create Account object
11       Account account2 = new Account( -7.53 ); // create Account object
12
13       // display initial balance of each object
14       System.out.printf( "account1 balance: $%.2f\n",
15          account1.getBalance() );
16       System.out.printf( "account2 balance: $%.2f\n\n",
17          account2.getBalance() );
18
19       // create Scanner to obtain input from command window
20       Scanner input = new Scanner( System.in );
21       double depositAmount; // deposit amount read from user
```

Output floating-point values with two-digits of precision

Fig. 3.14 | Inputting and outputting floating-point numbers with `Account` objects. (Part 1 of 3.)

AccountTest.java

```
22
23          System.out.print( "Enter deposit amount for account1: " ); // prompt
24          depositAmount = input.nextDouble(); // obtain user input
25          System.out.printf( "\nadding %.2f to account1 balance\n\n",
26             depositAmount );
27          account1.credit( depositAmount ); // add to account1 balance
28
29          // display balances
30          System.out.printf( "account1 balance: $%.2f\n",
31             account1.getBalance() );
32          System.out.printf( "account2 balance: $%.2f\n\n",
33             account2.getBalance() );
34
35          System.out.print( "Enter deposit amount for account2: " ); // prompt
36          depositAmount = input.nextDouble(); // obtain user input
37          System.out.printf( "\nadding %.2f to account2 balance\n\n",
38             depositAmount );
39          account2.credit( depositAmount ); // add to account2 balance
40
41          // display balances
42          System.out.printf( "account1 balance: $%.2f\n",
43             account1.getBalance() );
```

Returns a `double` value typed by the user

Fig. 3.14 | Inputting and outputting floating-point numbers with `Account` objects. (Part 2 of 3.)

AccountTest.java

```
44          System.out.printf( "account2 balance: $%.2f\n",
45             account2.getBalance() );
46       } // end main
47    } // end class AccountTest
```

```
account1 balance: $50.00
account2 balance: $0.00

Enter deposit amount for account1: 25.53

adding 25.53 to account1 balance

account1 balance: $75.53
account2 balance: $0.00

Enter deposit amount for account2: 123.45

adding 123.45 to account2 balance

account1 balance: $75.53
account2 balance: $123.45
```

Fig. 3.14 | Inputting and outputting floating-point numbers with `Account` objects. (Part 3 of 3.)

Recap

This lecture, we have learnt about:

- **Class**
  - **Attributes**
  - **Method** with Parameters
    - **Getters** and **Setters** Methods
  - **Access Modifiers** (Public & Private)
  - **Constructor** with Parameters
  - **UML** Diagram
- **Object**
  - **Create objects** from class with keyword **new**
  - Call **methods** with input arguments of the object
- **Primitive** vs **Reference**
- **Useful Classes**
  - **Scanner**
    - Read input string with nextLine(), next()
    - Read input number with nextDouble()
  - **String**
    - Display string with print(), println(), printf()
  - **Math**
    - pow(), max(), random()
- Bank account application example

Thank you for your listening!

“One who never asks
Either knows everything or nothing”

Malcolm S. Forbes