JAVA Classes and Constructors (more)

DR. KRISHNENDU GUHA
ASSISTANT PROFESSOR/ LECTURER
SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY COLLEGE CORK

Multiple Classes in a Single Java Program

- A single Java program contains two or more classes
- Two ways to implement
  - Nested Classes
  - Multiple non-nested classes

Multiple Non-nested classes

```
public class Computer {
		Computer() {
			System.out.println("Constructor of Computer class.");
		}
		void computer_method() {
			System.out.println("Power gone! Shut down your PC soon...");
		}
		public static void main(String[] args) {
			Computer c = new Computer();
			Laptop l = new Laptop();
			c.computer_method();
			l.laptop_method();
		}
}


		class Laptop {
			Laptop() {
				System.out.println("Constructor of Laptop class.");
			}
			void laptop_method() {
				System.out.println("99% Battery available.");
			}
```

the java program contains two classes, one class name is Computer and another is Laptop.

Both classes have their own constructors and a method.

In the main method, we can create an object of two classes and call their methods.

How Compiler Behaves

Output

Constructor of Computer class.
Constructor of Laptop class.
Power gone! Shut down your PC soon...
99% Battery available.

- When we compile the above program,
  - two **.class files** will be created which are ***Computer.class and Laptop.class.***
  - This has the **advantage** that ***we can reuse our .class file somewhere in other projects*** **without compiling the code again**.
  - In short, the **number of .class files created will be equal to the number of classes in the code**.
  - We can create as many classes as we want **but writing many classes in a single file is not recommended** as it **makes code difficult to read rather we can create a single file for every class**.

Nested Classes

```
// Main class
public class Main {
        class Test1 { // Inner class Test1    }
        class Test2 { // Inner class Test2    }

        public static void main(String [] args) {
                new Object() { // Anonymous inner class 1      };
                new Object() { // Anonymous inner class 2      };
                System.out.println("Welcome to Java");
        }
}
```

Once the **main class is compiled** which has several inner classes, the compiler **generates separate .class files for each of the inner classes**.

Output

Welcome to Java

In the above program, we have a Main class that has four inner classes **Test1, Test2, Anonymous inner class 1** and **Anonymous inner class 2**. Once we compile this class, it will generate the following class files.

- Main.class
- Main$Test1.class
- Main$Test2.class
- Main$1.class
- Main$2.class

more than one public class in a Java package?

- **No**,
- while defining multiple classes in a single Java file you need to make sure that **only one class among them is public**.
- If you have more than one public classes a single file a **compile-time error will be generated**.

Example

```
import java.util.Scanner;
public class Student {
		private String name;
		private int age;

		Student(){
			this.name = "R";
			this.age = 29;
		}
		Student(String name, int age){
			this.name = name;
			this.age = age;
		}
		public void display() {
			System.out.println("name: "+this.name);
			System.out.println("age: "+this.age);
		}
}

public class AccessData{
		public static void main(String args[]) {
			//Reading values from user
			Scanner sc = new Scanner(System.in);
			System.out.println("Enter the name of the student: ");
			String name = sc.nextLine();
			System.out.println("Enter the age of the student: ");
			int age = sc.nextInt();
			Student obj1 = new Student(name, age);
			obj1.display();
			Student obj2 = new Student();
			obj2.display();
		}
}
```

Compile-time error

On compiling, the above program generates the following compile-time error.

AccessData.java:2: error: class Student is public, should be declared in a file named Student.java public class Student { ^ 1 error

To **resolve** this **either**

- **you need to shift one of the classes into a separate file**
- or,

•Remove the public declaration before the class that doesn't contain a *public static void main(String args)* method.

•Name the file with the class name that contains main method.

In this case, remove the public before the Student class. Name the file as *"AccessData.java"*.

Java Constructors

- A constructor in Java is a **special method** that is used to initialize objects.
- The constructor is called when an object of a class is created.
- It can be used to set initial values for object attributes:

```
// Creating a Car class
public class Car
{
        int max_speed;          // Creating a class attribute
        // Creating a class constructor for the Car class
        public Car()
        {
                max_speed = 200; // Setting the initial value for the class attribute max_speed
        }
        public static void main(String[] args)
        {
                Car Toyota = new Car(); // Creating an object of class Car (This will call the constructor)
                System.out.println(Toyota.max_speed); // Prints the value of max_speed
        }
}
// Outputs 200
```

- Note that the constructor name must **match the class name**, and it cannot have a **return type** (like void).
- Also note that the constructor is called when the object is created.
- All classes have constructors by default: if you do not create a class constructor, Java creates one for you.
- However, then you are not able to set initial values for object attributes.

```
//Generic Example: Creating a Main class
public class Main
{
        int x;          // Creating a class attribute
        // Creating a class constructor for the Main class
        public Main()
        {
                x = 5; // Setting the initial value for the class attribute x
        }
        public static void main(String[] args)
        {
                Main myObj = new Main(); // Creating an object of class Main (This will call the constructor)
                System.out.println(myObj.x); // Prints the value of x
        }
}
// Outputs 5
```

Constructor Parameters

Constructors can also take parameters, which is used to initialize attributes.
The following example adds an int y parameter to the constructor.
Inside the constructor we set x to y (x=y).
When we call the constructor, we pass a parameter to the constructor (5), which will set the value of x to 5:

```
public class Main
{
        int x;
        public Main(int y)
        {
                x = y;
        }
        public static void main(String[] args)
        {
                Main myObj = new Main(5);
                System.out.println(myObj.x);
        }
}
// Outputs 5
```

- You can have as many parameters as you want:

```
public class Car
{
        int modelYear;
        String modelName;
        public Car(int year, String name)
        {
                modelYear = year;
                modelName = name;
        }
        public static void main(String[] args)
        {
                Car Toyota = new Car(1969, “UCC");
                System.out.println(Toyota.modelYear + " " + Toyota.modelName);
        }
}
// Outputs 1969 UCC
```

Java Modifiers

```
public class Main
```

The public keyword is an **access modifier**,
meaning that **it is used to set the access level for classes, attributes, methods and constructors.**

Modifiers can be divided into two groups:
•**Access Modifiers** - controls the access level
•**Non-Access Modifiers** - do not control access level, but provides other functionality

Access Modifiers
For **classes**, you can use either public or *default*:

Public: The class is accessible by any other class

Default: The class is only accessible by classes in the same package.
This is used when we do not specify a modifier

- For **attributes, methods and constructors**, you can use the one of the following:

- **Public**: the code is *accessible to all classes*
- **Default**: The code is only accessible in the *same package*.
  - This is used when we do not specify a modifier

- **Private**: The code is only *accessible within the declared class*
- **Protected**: the code is accessible in the *same package and subclasses*

Non-Access Modifiers
For **classes**, we use either final or abstract:

Final: The class cannot be inherited by other classes

Abstract: The class cannot be used to create objects
To access an abstract class, it must be inherited from another class

- For **attributes and methods**, you can use the one of the following:
- **Final**: Attributes and methods *cannot be overridden or modified*
- **Static**: Attributes and methods *belongs to the class, rather than an object*
- **Abstract**: Can *only be used in an abstract class*, and can *only be used on methods*

  The method does not have a body, example: abstract void run()

  The body is provided by the subclass (inherited from)

- **Transient**: *Attributes and methods are skipped* when serializing the object containing them
- **Synchronized**: Methods can *only be accessed by one thread at a time*
- **Volatile**: The value of an attribute is *not cached locally and is read from the "main memory"*

Final

If you don't want the ability to override existing attribute values, declare attributes as final:

Example

```
public class Car
{
        final int max_speed = 100;
        final double fuel_capacity = 100.14;
        public static void main(String[] args)
        {
                Car Toyota = new Car();
                Toyota.max_speed = 150; // will generate an error: cannot assign a value to a final variable
                Toyota.fuel_capacity = 125.5; // will generate an error: cannot assign a value to a final variable
                System.out.println(Toyota.max_speed);
        }
}
```

Static: A static method means that it can be accessed without creating an object of the class, unlike public

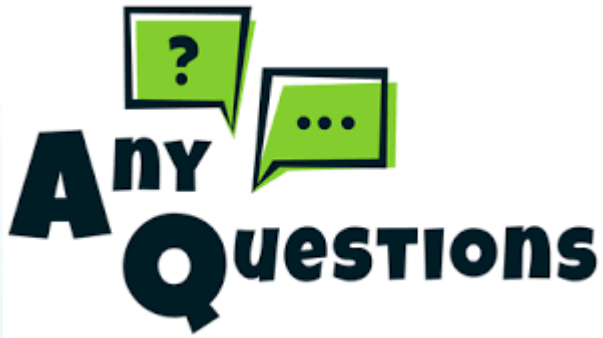
An example to demonstrate the differences between static and public methods:

```
public class Main
{
        // Static method
        static void myStaticMethod()
        {
                System.out.println("Static methods can be called without creating objects");
        }
        // Public method
        public void myPublicMethod()
        {
                System.out.println("Public methods must be called by creating objects");
        }
        // Main method
        public static void main(String[ ] args)
        {
                myStaticMethod(); // Call the static method
                // myPublicMethod(); This would output an error
                Main myObj = new Main(); // Create an object of Main
                myObj.myPublicMethod(); // Call the public method
        }
}
```

Abstract: An abstract method belongs to an abstract class, and it does not have a body.
The body is provided by the subclass:

```
// Code from filename: Main.java
// abstract class
abstract class Main {
        public String fname = "John";
        public int age = 24;
        public abstract void study(); // abstract method
        }
// Subclass (inherit from Main)
class Student extends Main {
        public int graduationYear = 2018;
        public void study()
        {
                // the body of the abstract method is provided here
                System.out.println("Studying all day long");
        }
} // End code from filename: Main.java

// Code from filename: Second.java
class Second {
        public static void main(String[] args) {
                // create an object of the Student class (which inherits attributes and methods from Main)
                Student myObj = new Student();
                System.out.println("Name: " + myObj.fname);
                System.out.println("Age: " + myObj.age);
                System.out.println("Graduation Year: " + myObj.graduationYear);
                myObj.study(); // call abstract method
        }
}
```


Any Questions


Thank You!