+

Computer programming (1)

Chapter 2

Introduction to Classes and Objects (1)

Agenda

- Overview
- Introduction to Object Oriented Programming (OOP)
  - Features of Object-Oriented Languages
- Introduction to classes and objects
- Classes

Overview

- Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems in 1991.
- This language was initially called "Oak" but was renamed "Java" in 1995.
- The fundamental forces that necessitated the invention of Java are:
  - Portability
  - Security

Overview

- The key that allows Java to solve both the security and the portability problems just described is that the output of a Java compiler <u>is not executable code</u>. Rather, it <u>is bytecode</u>.

- *<u>Bytecode</u>* is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the *Java Virtual Machine (JVM).*

Overview

- Translating a Java program into bytecode makes it much easier to run a program in a wide variety of environments because only the JVM needs to be implemented for each platform.
- The fact that a Java program is executed by the JVM also helps to make it secure. Because the JVM is in control, it can contain the program and prevent it from generating side effects outside of the system.

Agenda

- Overview
- Introduction to Object Oriented Programming (OOP)
  - Features of Object-Oriented Languages
- Introduction to classes and objects
- Classes

+

Object-Oriented Programming

- It is useful to understand OOP's basic principles before you write even a simple Java program.
- OOP is a powerful way to approach the job of programming
- All OOP languages, including Java, have three traits in common:
  - Encapsulation
  - Inheritance
  - Polymorphism

+

Encapsulation in Java

- *Encapsulation* is a programming mechanism that binds together code and the data it manipulates, and that keeps both safe from outside interference and misuse.
- Wrapping up data member and method together into a single unit (i.e. Class) is called **Encapsulation.**
- When code and data are linked together in this fashion, an object is created

+

Agenda

- Overview
- Introduction to Object Oriented Programming (OOP)
  - Features of Object-Oriented Languages
- Introduction to classes and objects
- Classes

Concepts of Classes and Objects

- Java's basic unit of encapsulation is the *class*.
- A **class** can be defined as a template/blue print that describes the behaviors/states that object of its type support.
- A class defines the form of an object. It specifies both the data and the code that will operate on that data. Java uses a class specification to construct *objects.*
- Objects are instances of a class. **Objects have states and behaviors.**

➔Thus, a class is essentially a set of plans that specify how to build an object.

+

Classes in Java:

- **Class = data + methods.** Everything (data and methods) in Java is contained in *classes*
- A class is a blueprint from which individual objects are created.

Anatomy of a Java Class

Class AccessLevel   Keyword "Class"   Class Name

$$\begin{tikzcd}
\text{Class AccessLevel} \arrow[d] & \text{Keyword "Class"} \arrow[d] & \text{Class Name} \arrow[d] \\
\text{public} & \text{class} & \text{ClassName \{}
\end{tikzcd}$$

```
public          class          ClassName  {

   // Variables  ←these represents class attributes

   // Methods ←these represents class behavior

}
```

+

Class

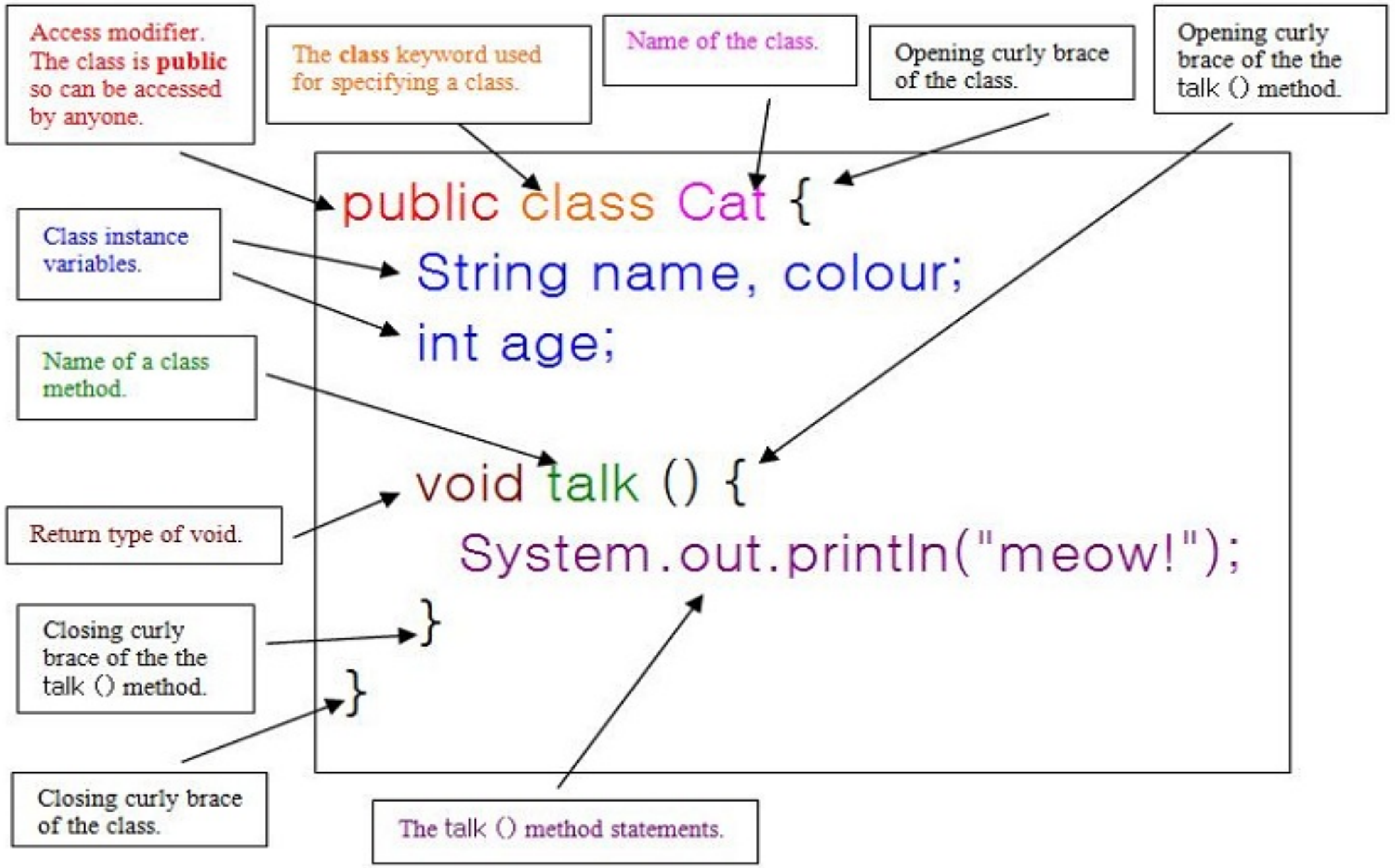
- A class is the core(heart) of any modern Object Oriented Programming language
- In OOP languages, it is a must to create a class for representing data.
- Class will not occupy any memory space and hence it is only a logical representation of data.
- To create a class, you simply use the keyword "class" followed by the class name:

```
class Employee
  {
        class members
  }
```

Class Members

- *fields*: data variables which determine the status of the class or an object
- *methods*: executable code of the class built from statements. It allows us to manipulate/change the status of an object or access the value of the data member

Defining A Class



```
public class Cat {
    String name, colour;
    int age;

    void talk () {
      System.out.println("meow!");
    }
}
```

Access modifier. The class is **public** so can be accessed by anyone.
The **class** keyword used for specifying a class.
Name of the class.
Opening curly brace of the class.
Opening curly brace of the the talk () method.
Class instance variables.
Name of a class method.
Return type of void.
Closing curly brace of the the talk () method.
Closing curly brace of the class.
The talk () method statements.

+

Example

```
Class Account {
    String clientName;
    String accountNo;
    double balance;

    }
```

Instance Variables

Variable types

- A class can contain any of the following variable types.
- **Local variables:** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables:** Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables:** Class variables are variables declared with in a class, outside any method, with the static keyword.

+
Terminology

- *Instance variable* = non static variable declared in class outside a method.
- Class variable = static variable declared in class outside a method.
- *Field* = Synonym for *instance variable*, *attribute*. Common informal term.
- *Attribute* = Synonym for *instance variable*, *field*. Often used in the design phase.

Quick Check

1. Write Text-Based Application using Object-Oriented Approach to display your name.

Quick Check

```
// filename: Name.java

// Class containing display() method, notice the class doesn't have
a main() method

public class Name {

    String name;

        public void display() {

                System.out.println(name);

                    }

}
```

+

Quick Check

```
// filename: DisplayName.java

// place in same folder as the Name.java file

// Class containing the main() method

public class DisplayName {

        public static void main(String[] args) {

        // creating a new object of Name class

        Name myname = new Name();

    myname.name = "Mohamed";

        // executing the display() method in the Name class

        myname.display();

        }

}
```

Exercise

- A class called **Circle** contains:
  - Two instance variables (attributes): radius (typed double) and color (typed String).
  - Two public methods: **getRadius**() and **getArea**(), which return the radius and area of this instance, respectively.
- Create another class called **TestCircle** to :
  - Create two different objects "**Circle**",
  - Give values of attributes
  - Call methods

+

Thanks!