Introduction to OOP

Vũ Thị Hồng Nhạn

(vthnhan@vnu.edu.vn)

Department of software engineering

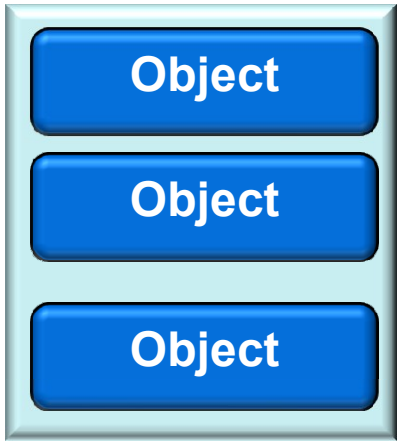
Vietnam National Univ., Hanoi

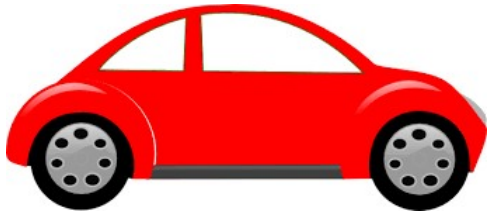
Contents

- What is OOP
- Object vs. class
- Fields, methods
- method main()

What is OOP?

- OPP is a type of programming that is driven by modeling your code around **building** units called **objects**
  - Each object, as its name implies, represents a real object in the world around us *like a person, a table, a building, a book, a car, a tree*..





OOP languages

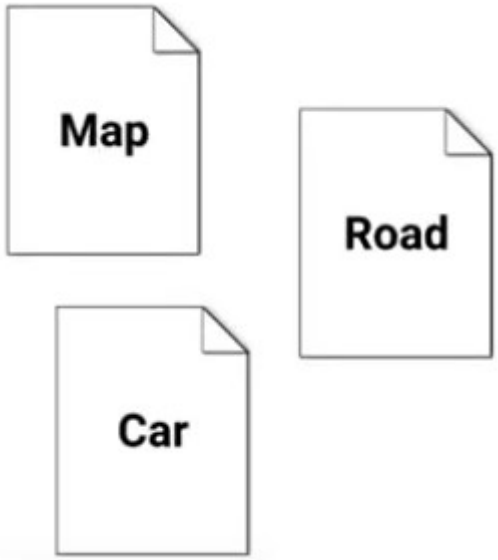
- Nowadays, almost every **modern programming language** you've heard of **is object-oriented** including Java

idea of oop

- When you are coding you want to solve a real world problem and modeling your code to match what you're trying to solve makes perfect sense

Real world

Java



Example

- Build a Pokemon game in java

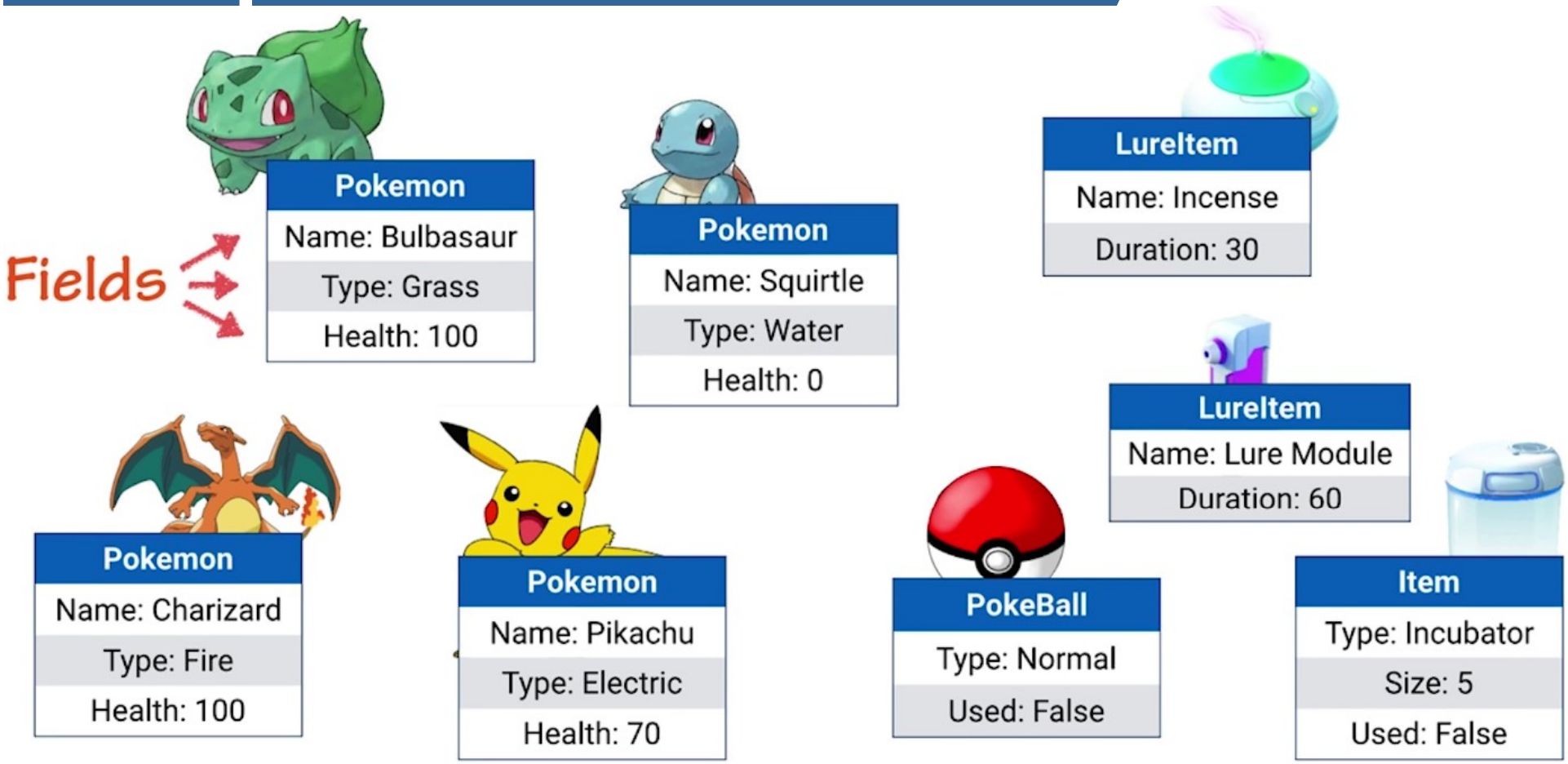
Example

- Build a Pokemon game in java

- Pokemon objects & items

•Need to have **an object** for **each** of the *Pokemon characters*, an object for *each item* he can carry

Example…



- Each object is responsible for holding the data that describes itself
- referred as FIELDs

Example…

Pokemon
Name: Pikachu
Type: Electric
Health: 70
attack()
dodge()
evolve()

Fields

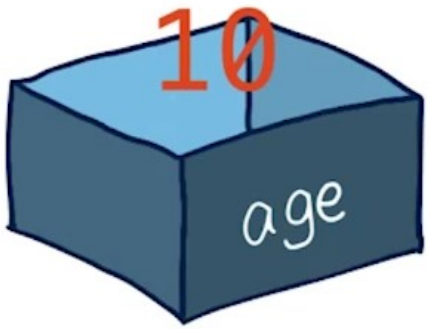
Methods

- Along having **fields**, objects are also capable of performing **actions**
- e.g., Pokemon can *attack, dodge,* and *evolve*
- These actions in java are referred to as **methods**

Variable types

- Just like creating **variables of basic datatypes** like integers and doubles (known as **primitive variables)**
- **An object** is nothing more than an enhanced datatype that you get to design yourself

Primitive variables	Object variables
int age; double score;	Pokemon pikachu; Item incubator;



Why use objects?

- Objects combine variables together to make your code meaningful
- **code** becomes more organized and easier to understand
- Maintaining your code becomes much simpler
- **java won't work without objects**

Fields

- Fields of an object
  - are all the data variables that makes up that object
  - sometimes are referred to as attributes or member variables
- **Fields** are usually made up of
  - primitive types (integer, character…)
  - but can also be objects themselves
- E.g., a book object may contain **fields** like…

```
String title;
String author;
int numberOfPages;
```

  - Then, a library object may contain a field named books that will store all book objects *in an array*

Fields…

- Accessing a field in an object is done using the dot modifier “.”
- e.g., to access the title field of an object called “book” you would use book.title
- You can use it directly as primitive variable and perform operations
  - String bookTitle = book.title; //store in a string variable
  - System.out.println(book.tititle); // printing
- you can change a field’s value
  - book.numberOfpages=100;

Methods

- Running **actions** in objects look very much like calling a function
- Methods in java are functions that belong to a particular object
- To call a method in an object we use the dot modifier **“.”**
- E.g.
  - Assume a book **object** has a method called setBookmark(int pageNo) that takes the page number as *parameter*
  - if you want **to set** a bookmark at page 30, you can call the method and pass in the page number as an argument
  - book.setBookmartk(30);

summary

- **Fields** and **methods** make an object useful
  - **Fields** store the object's data
  - while **methods** perform actions to use or modify those data
- **However, some objects** might have no fields and are just made up of a bunch of methods that perform various actions
- **Other objects** might only have fields that act as a way to organize storing data but not include any methods

Integrated development environment (IDE)

- Java simply are plain text files with extension .java
- **Java project** is a folder that contains a bunch of those java files

- You can create a java project with a basic text editor
- and need a compiler that runs on the command line

- But how to make the development experience **more pleasant ?**

IDE…

- To be able to create and run any code in Java, you need two main things
  - A helpful text editor that highlights keywords with different colors and auto complete code
  - **a compiler** that converts your java code into computer code (known as bytecode) that can be understood by computers and hence run properly

- An IDE combines both of those

IDE…

- There are plenty of options out there, choosing one is usually based on the programming language you're using and your personal preference

- A list of the most commonly used Java IDEs
  - Eclipse
  - Intellij
  - Android studio (based on Intellij)
  - NetBeans
  - BlueJ

Class vs. Object

- To design objects, we need to create **classes**
- Class can be seen as the blueprint that defines what object should look like
- An object on the other hand is the actual entity that is created from that class
- i.e., **a class** is where you would list all the fields and implement all the methods when defining what that object type should look like

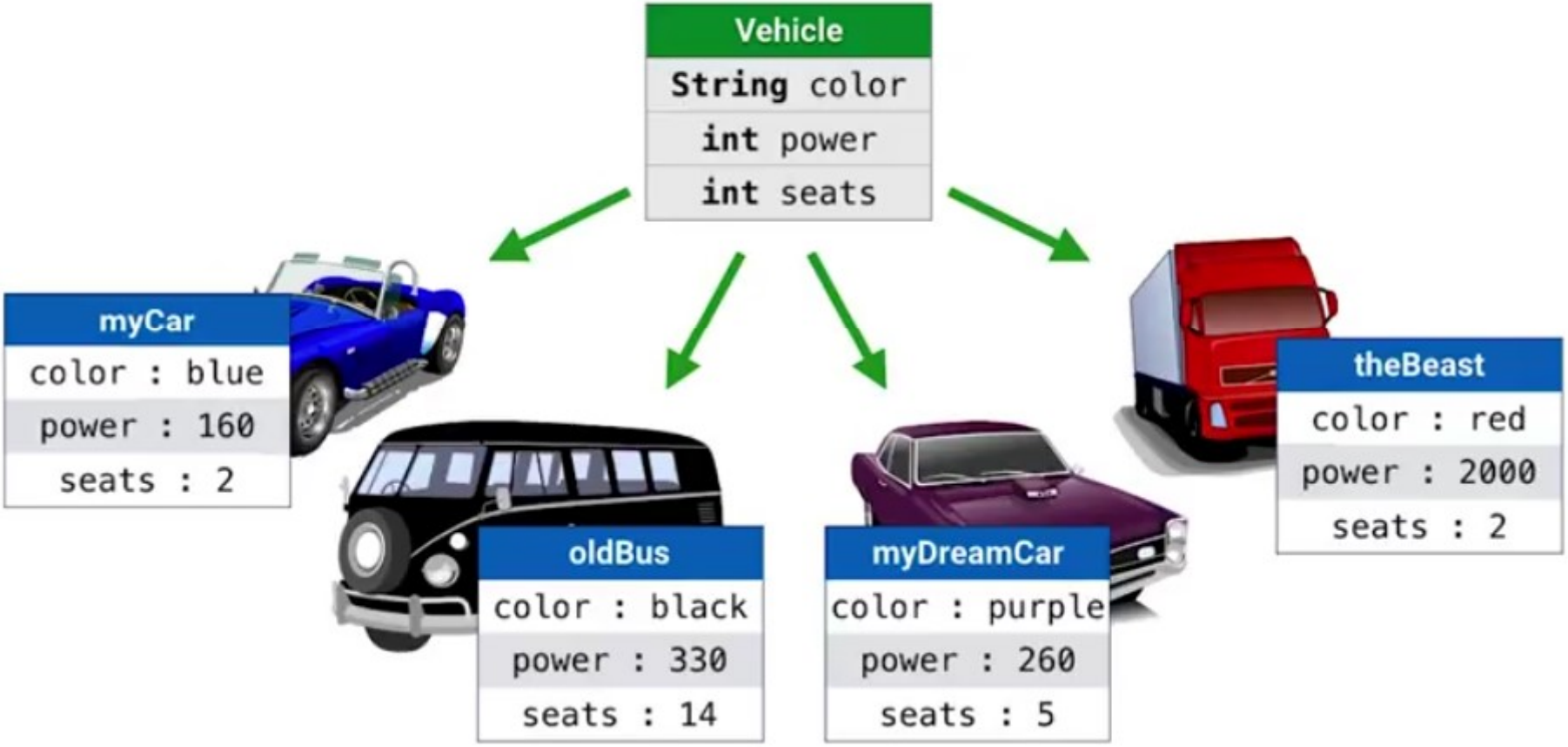
Example 1

```
class Pokemon{

    String name;
    String type;
    int health;

    boolean dodge(){
        return Math.random() >0.5;
    }
    void attack(Pokemon enemy){
        if(!enemy. doge())
            enemy.health--;
    }
}
```

fields

methods

Example 2


Vehicle
String color
int power
int seats
myCar
color : blue
power : 160
seats : 2
oldBus
color : black
power : 330
seats : 14
myDreamCar
color : purple
power : 260
seats : 5
theBeast
color : red
power : 2000
seats : 2

Class

- in Java, **each class** should be created in its own file
- No java code can live anywhere outside a class
- Classes (java files) interact with each other

```java
class Vehicle {

  String color;
  int power;
  int seats;

}
```

Vehicle.java

```java
class Map {

  String city;
  double lan;
  double lon;

}
```

Map.java

```java
class Road {

  String name;
  int speed;
  double length;

}
```

Road.java

Classes vs. Objects

	Class	Object
What:	A Data Type	A Variable
Where:	Has its own file	Scattered around the project
Naming convention:	CamelCase (starts with an upper case)	camelCase (starts with a lower case)
Examples:	Country	australia
	Book	lordOfTheRings
	Pokemon	pikachu

String

- is not a primitive type, but a class
  - starts with upper case ‘S’
- **a String variable**
  - is made up of an array of characters (char [])
  - being **an object** means that it also offers **some methods** like length() that counts the number of characters in that array
  - or equal(String s) that compares the characters in this string with another string

Everything is an object in Java

- Because java is an OOP language, it includes classes that simply wrap around all the primitive types themselves to offer some extra functionality through their methods

Class	Primitive type
Integer	int
Long	long
Double	double
Character	char
String	char[]

- Each of these classes is made up of the corresponding primitive types as its fields, but usually also comes with some powerful methods

The main() method

- **A java program** can be as small as a single class
  - but usually a single program will be made up of 10 or even 100 of classes
- **A good java program** is one that divides the logic appropriately so that each class ends up containing everything related to that class and nothing more
- Classes would **call** each other's methods and **update** their fields to make up the logic of the entire program all together
- But, where should the program start from exactly?
  - the answer is the main() method

The main() method…

```
public static void main(String [] args){
    //start my program here
}
```

- public: means you can run this method from anywhere in your java program
- static: means it doesn't need an object to run, which is why the computer starts with this method before even creating any objects
- void: the main method doesn't return anything, it just runs when the program starts and once it's done the program terminates
- String [] args: is the input parameter (array of Strings)

The main() method…

- This method is the starting point for any Java program, when a computer runs a Java program, it looks for that main() method and runs it
- inside it, you can create objects and call methods to run other parts of your code
- The **main** method can belong to **any class**, or you can create a specific class just for that **main method** which is what most people do

Creating a class

- An example: open Notepad and edit the text below

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello Java");
    }
}
```

- Save in HelloWorld.java
- Compile and Run with **Command Window**

Creating a class...

Command Prompt

```
Microsoft Windows [Version 10.0.17134.48]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\vthnhan>cd Desktop

C:\Users\vthnhan\Desktop>javac HelloWorld.java

C:\Users\vthnhan\Desktop>java HelloWorld
Hello Java

C:\Users\vthnhan\Desktop>
```

Compile

Run