Introduction to Object Oriented Programming

Java™ How to Program, 10/e
Late Objects Version

References & Reading

- The content is mainly selected (sometimes modified) from the original slides provided by the authors of the textbook

- Readings
  - Chapter 1: Introduction to Computers, the Internet and Java
  - Chapter 6: Arrays and ArrayLists

Outline

1.5 Introduction to Object Technology

6.2 Primitive Types vs. Reference Types

1.5 Introduction to Object Technology

- Objects, or more precisely, the *classes* objects come from, are essentially *reusable* software components.
  - There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc.
  - Almost any *noun* can be reasonably represented as a software object in terms of *attributes* (e.g., name, color and size) and *behaviors* (e.g., calculating, moving and communicating).
- Software development groups can use a modular, object-oriented design-and-implementation approach to be much more productive than with earlier popular techniques like “structured programming”—object-oriented programs are often easier to understand, correct and modify.

Problem

- Write a java program to store personal information include: name, gender, job, and age for 4 person.

...

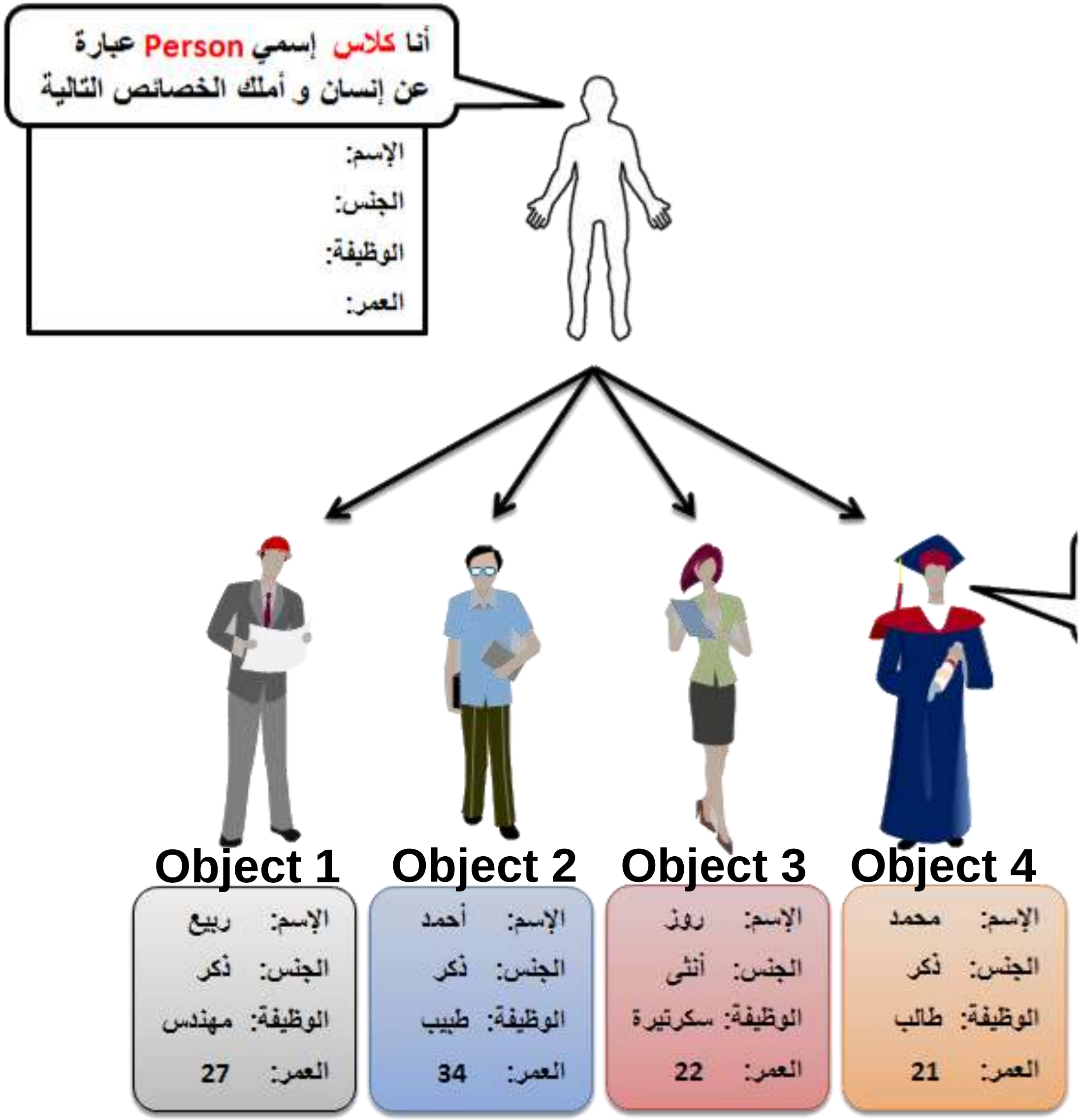
```
String name1= “محمد”;
String gender1 = “ذكر”;
String job1= “طالب”;
int age= 21;

String name2= “روز”;
String gender2 = “أنثى”;
String job2= “سكرتيرة”;
int age2= 22;

String name3= أحمد“”;
String gender3 = “ذكر”;
String job3= “طبيب”;
int age3 = 34;

String name4= “ربيع”;
String gender4= “ذكر”;
String job4= “مهندس”;
int age4= 27;
```

الإسم: محمد
الجنس: ذكر
الوظيفة: طالب
العمر: 21

الإسم: روز
الجنس: أنثى
الوظيفة: سكرتيرة
العمر: 22

الإسم: أحمد
الجنس: ذكر
الوظيفة: طبيب
العمر: 34

الإسم: ربيع
الجنس: ذكر
الوظيفة: مهندس
العمر: 27



أنا كلاس إسمي Person عبارة عن إنسان و أملك الخصائص التالية

الإسم:

الجنس:

الوظيفة:

العمر:

Object 1	Object 2	Object 3	Object 4
الإسم: ربيع	الإسم: أحمد	الإسم: روز	الإسم: محمد
الجنس: ذكر	الجنس: ذكر	الجنس: أنثى	الجنس: ذكر
الوظيفة: مهندس	الوظيفة: طبيب	الوظيفة: سكرتيرة	الوظيفة: طالب
العمر: 27	العمر: 34	العمر: 22	العمر: 21

Solution

- Notes:

1. A class has empty *attributes*.

2. An object is a copy (*instance*) of a class.

- Each object has a copy of *attributes* and methods (*behaviors*)

3. Each object fills its own *attributes*.

4. We can handle each object separately.

5. Create a class firstly. Then generate its objects.

6. To add a new *attribute*, we declare it inside a class. class

   - Then it will be copied to all objects automatically.

7. We can imagine that a class is your data-type and an object is a variable of that type.

1.5.2 Methods and Classes

- In Java, we create a program unit called a class to house the set of methods that perform the class’s tasks.
- Performing a task in a program requires a method.
- The method houses the program statements that actually perform its tasks and hides these statements from its user
- Example of class: `Account`
- Example of methods: `withdraw()` and `deposit()`

1.5.3 Instantiation

- Just as someone has to *build* a car from its engineering drawings before you can actually drive a car, you must *build an object* of a class before a program can perform the tasks that the class's methods define.
- An object is then referred to as an instance of its class.

- Example of object: `Account account1`;
  - `Account` is the class
  - `account1` is an instance of `Account`

1.5.4 Reuse

- Just as a car's engineering drawings can be *reused* many times to build many cars, you can reuse a class many times to build many objects.
- Reuse of existing classes when building new classes and programs saves time and effort.
- Reuse also helps you build more reliable and effective systems, because existing classes and components often have undergone extensive *testing*, *debugging* and *performance* tuning.

1.5.6 Attributes and Instance Variables

- A car has *attributes*
- Color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading).
- The car’s attributes are represented as part of its design in its engineering diagrams.
- Every car maintains its *own* attributes.
- Each car knows how much gas is in its own gas tank, but *not* how much is in the tanks of *other* cars.

1.5.6 Attributes and Instance Variables (Cont.)

- An object, has attributes that it carries along as it’s used in a program.
- An `Account` object has a `balance` *attribute* that represents the amount of money in the account.
- Each `Account` object knows the balance in the account it represents, but *not* the balances of the *other* accounts in the bank.
- Attributes are specified by the class’s instance variables.

1.5.7 Encapsulation

- Classes (and their objects) encapsulate, i.e., encase, their attributes and methods.
- Objects may communicate with one another, but they're normally not allowed to know how other objects are implemented—implementation details are *hidden* within the objects themselves.
- Information hiding, as we'll see, is crucial to good software engineering.

1.5.8 Inheritance

- A new class of objects can be created conveniently by inheritance—the new class (called the subclass) starts with the characteristics of an existing class (called the superclass), possibly customizing them and adding unique characteristics of its own.

- Example: an object of class “convertible” certainly *is an* object of the more *general* class “automobile”, but more *specifically*, the roof can be raised or lowered.

1.5.9 Interfaces

- Interfaces are collections of related methods that typically enable you to tell objects *what* to do, but not *how* to do it

- This feature allows programmers to work similarly with different APIs since they implement the same interface(s).

- A class implements zero or more interfaces, each of which can have one or more methods, just as a car implements separate interfaces for basic driving functions, controlling the radio, controlling the heating and air conditioning systems, and the like.

1.5.10 Object-Oriented Analysis and Design (OOAD)

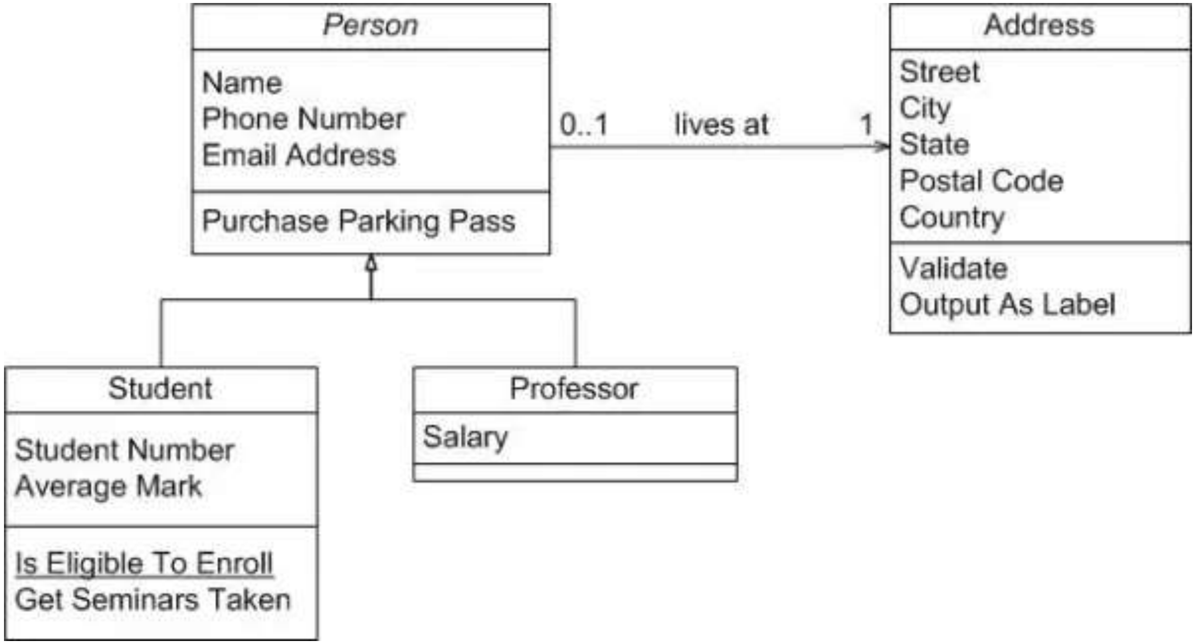
- How will you create the code (i.e., the program instructions) for your programs?
- Follow a detailed analysis process for determining your project’s requirements (i.e., defining *what* the system is supposed to do)
- Develop a design that satisfies them (i.e., specifying *how* the system should do it).
- Carefully review the design (and have your design reviewed by other software professionals) before writing any code.

1.5.10 Object-Oriented Analysis and Design (OOAD) (Cont.)

- Analyzing and designing your system from an object-oriented point of view is called an object-oriented-analysis-and-design (OOAD) process.
- Languages like Java are object oriented.
- Object-oriented programming (OOP) allows you to implement an object-oriented design as a working system.

1.5.11 The UML (Unified Modeling Language)

- The Unified Modeling Language (UML) is the most widely used graphical scheme for modeling object-oriented systems.



Person

Name
Phone Number
Email Address

Purchase Parking Pass

0..1 lives at 1

Address

Street
City
State
Postal Code
Country

Validate
Output As Label

Student

Student Number
Average Mark

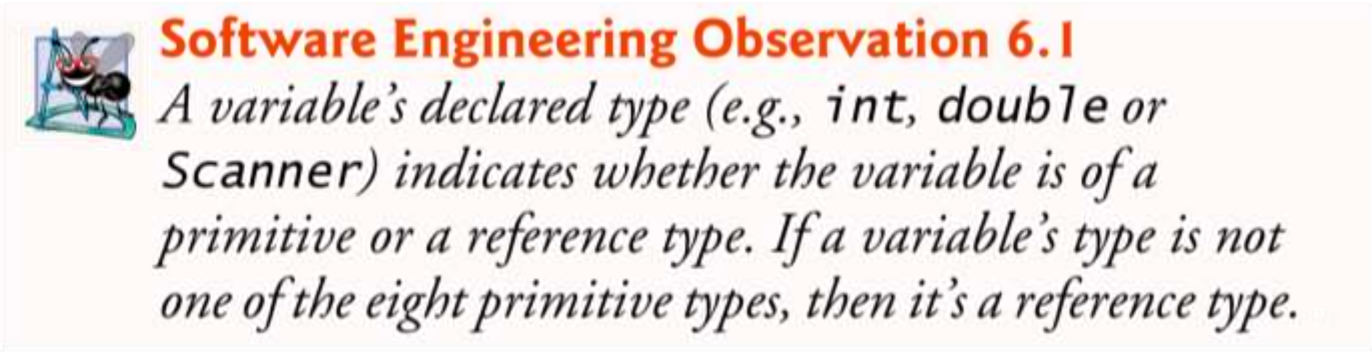
Is Eligible To Enroll
Get Seminars Taken

Professor

Salary

6.2 Primitive Types vs. Reference Types

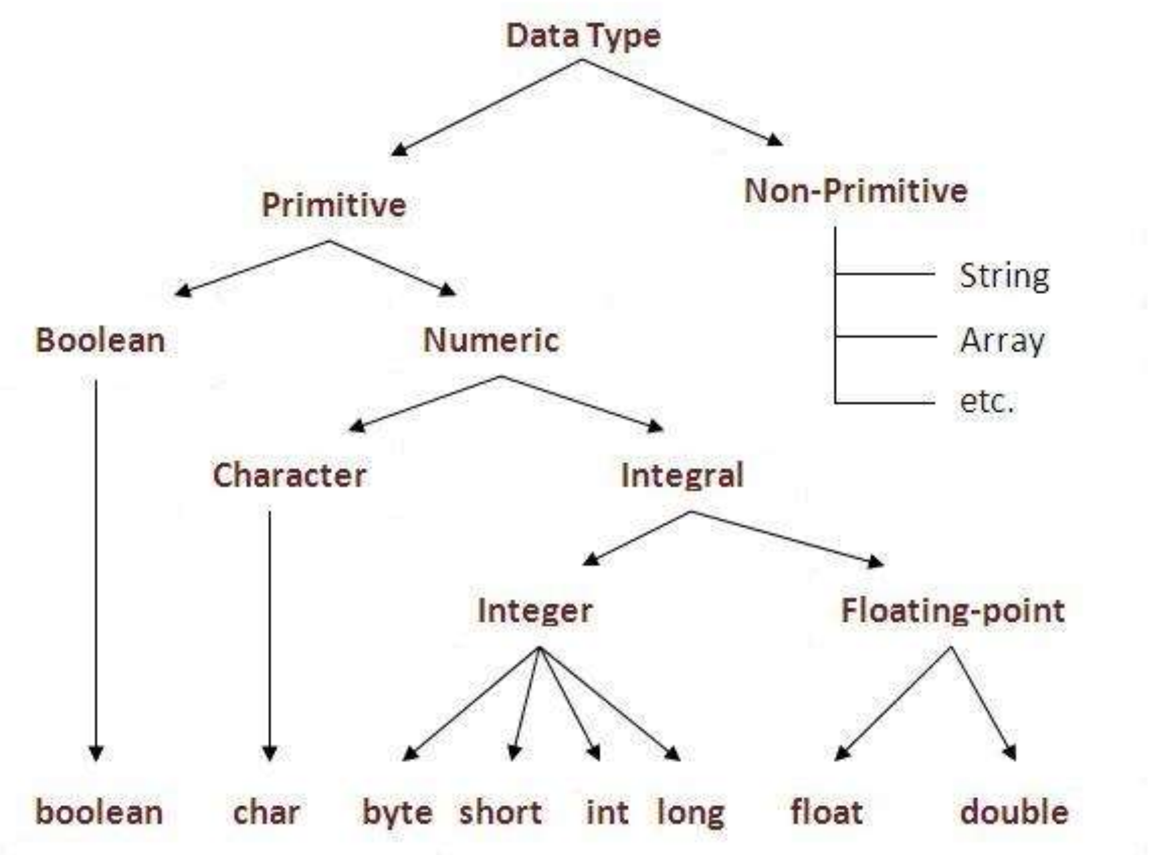
- Java’s types are divided into primitive types and reference types.
- Primitive types: `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` and `double`.
  - Appendix D lists the eight primitive types in Java.
- All nonprimitive types (classes) are *reference types*.



Software Engineering Observation 6.1

A variable’s declared type (e.g., `int`, `double` *or* `Scanner`*) indicates whether the variable is of a primitive or a reference type. If a variable’s type is not one of the eight primitive types, then it’s a reference type.*

6.2 Primitive Types vs. Reference Types (cont.)



6.2 Primitive Types vs. Reference Types (cont.)

- A primitive-type variable can hold exactly *one* value of its declared type at a time.
- Programs use variables of reference types (normally called references) to store the *addresses* of objects in the computer's memory.
  - Such a variable is said to refer to an object in the program.
- To call an object's methods, you need a reference to the object.
- If an object's method requires additional data to perform its task, then you'd pass arguments in the method call.
- Primitive-type variables do *not* refer to objects, so such variables cannot be used to invoke methods.