Lecture 2: Java & Javadoc

CS 62

Spring 2018

Alexandra Papoutsaki & William Devanny

Methods

- A collection of grouped statements that perform a logical operation and control the behavior of objects
- Syntax:
  - modifier return-type method-name(type parameter-name,…)
  - e.g., `public int enrollInClass(int classID){…}`
  - *Signature:* method name and the number, type and order of its parameters
- Can also be `static`, therefore shared by all instances of a class
- Can be overloaded (same name, different parameters)

this

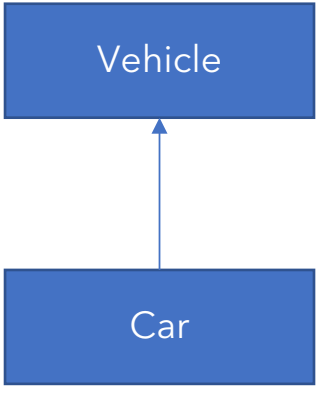
- Within an instance method or a constructor used to refer to current object
  - can be used to call instance variables, methods, and constructors

```
public class Car{
     private String color;

     public Car(){
          this(“undefined”);
     }
     public Car(String color){
          this.color = color;
     }
```

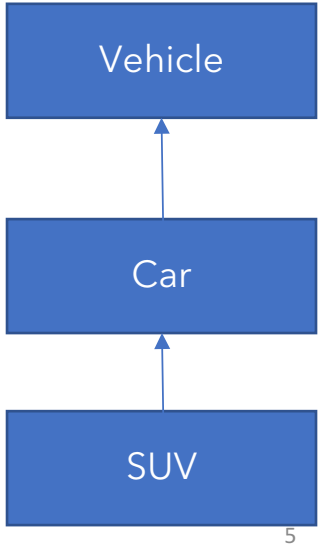
Inheritance

- When you want to create a new class and there is already a class that includes some of the code you want your new class to have, you can derive the new class from the existing class → reuse code!
- We say that a class *extends* or *inherits* another class
- E.g., `public class Car extends Vehicle`
- `Car` is a subclass of `Vehicle`
- `Vehicle` is a superclass of `Car`
- `Car` IS-A `Vehicle`



Inheritance in Java

- A subclass inherits all of the `public` and `protected` members of parent
- *Hiding:* same name of variables between super and subclass
- *Overriding:* same signature of methods between super and subclass
  - *Hiding* if static
- Single inheritance!
  - A class can only extend ONE AND ONLY ONE class
- Multilevel inheritance
  - Class SUV extends class `Car` which extends class `Vehicle`



`super` keyword

- refers to the direct parent class of the current class
- `super.variable` (for hidden fields → avoid altogether)
- `super.method()` (for overriden methods)
- `super(args)` → to call the constructor of the superclass

All classes inherit `Object`

- Directly (if they do not extend any other class) or indirectly
- Object class has methods (and more):
  - `public boolean equals (Object other)`
    - Default behavior returns true only if same object
  - `public String toString()`
    - Returns string representation of object – default is hexadecimal
    - Does not print the string
    - Typically needs to be overwritten to be useful
  - `public int hashCode()`
    - Unique identifier defined so that if `a.equals(b)` then a, b have same hashCode

`final`

- variable – only assigned once in its declaration or constructor – cannot change

- method – cannot be overriden by subclass
  - Methods called from constructors should generally be declared final

- class - cannot be extended

abstract

- Class – cannot be instantiated but can be extended
- Method – declared without an implementation
  - no braces and body, just semicolon
  - `public abstract int enrollInClass(int classID);`
- If a class has at least one abstract method then it should be declared abstract itself
- If you extend an abstract class either declare subclass as abstract too or implement the methods

Interfaces

- Contracts on how the program should work, abstracting from implementation
  - `public interface Moveable{…}`
- A class can *implement* many interfaces
  - `public class Car extends Vehicle implements Moveable`
- Variables - automatically `public`, `static`, and `final`
- Methods – `public` (declared or default)
- Cannot be instantiated

Nested class

- A class defined within a class, e.g., it's useful only within that one
- ```
  class Outer{
    …
    (static) class Inner{…}
    }
  ```
- Can be `static` or non-static (inner)
  ```

Enum Types

- Example
  - `enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES}`
- Operations:
  - `int compareTo(Suit other)`
  - `String toString()`
  - `int ordinal()` *starts with 0, not 1*
  - `static Suit valueOf(String name)`
  - `static Suit[] values()` *returns array of all values*

Documentation

- Important for code maintainability
  - This matters even for 1st week assignments
- Critical when working on a team
- Create documentation first– this is design work!

JavaDoc

- Document generation system
  - Reads JavaDoc comment →HTML pages
- JavaDoc comment = description written in HTML + tags
- Enclosed in `/**  */`
- Must precede class, variable, constructor or method declaration
- Read the style guide

JavaDoc

- Common tags:
  - for class:
    - **@author** author name – classes and interfaces
    - **@version** date - classes and interfaces
  - for method:
    - **@param** param name and description – methods and constructors
    - **@return** value returned, if any – methods
    - **@throws** description of any exceptions thrown - methods

Packages

- Use them! E.g., `package assignment1;` … before everything else
- Package name == folder name
- Helps organize large projects e.g, `java.lang`→fundamental
- Import a package member: `import package.member;`
- Import an entire package: `import package.*;`

Generics

- Enable classes and interfaces to be parameters when defining classes, interfaces, and methods.
  - `class Name<T1, T2, ..., Tn> {…}`
  - `T` can be used anywhere within the class
  - `T` can be any *non-primitive*
- `T` →Type, `E` →Element, `K`→ Key, `V`→Value, `N`→number
- See **`Association`** class in Bailey `structure5` library
  - `public class Association<K,V>`
  - `Association<String, Integer> phoneBook = new Association<String, Integer> ();`

Random Number Generator

- class **Random** in **java.util** package w/ method
  - `int nextInt(int n)` -- returns random k s.t. $0 \leq k < n$
  - See bottom of pg 30 in text.

- Create **Random** object once:
  - `Random rng = new Random();`
- send **nextInt** many times:
  - `int r = rng.nextInt(10);`
  - Repeat this step, not the creation of a new object

- See **LottoHelper** example.