Java Enterprise Application Development

Lecture 4
Classes and Objects

Dr. Fan Hongfei
24 September 2025

Declaring Classes

- Class declaration

```
class MyClass extends MySuperClass implements YourInterface {
      // field, constructor, and
      // method declarations
}
```

- The modifiers public and private

Declaring Member Variables

- Several kinds of variables
  - Member variables in a class: **fields**
  - Variables in a method or block: **local variables**
  - Variables in method declarations: **parameters**
- Field declarations are composed of
  - Zero or more modifiers, such as public or private
  - The field's type
  - The field's name

Defining Methods

- Method declarations have six components
  - Modifiers
  - The return type
  - The method name
  - The parameter list
  - An exception list
  - The method body

```
public int divide(int x, int y) throws DividedByZeroException {
        if ( y == 0 ) {
            throw new DividedByZeroException();
        }
        return x / y;
    }
```

- Naming a method: convention
- **Signature: method name + parameter types**
  - Example: *divide(int x, int y)*
- **Overloading methods: based on signature match**

Passing Information to a Method

- Passing **by value, at any time**
  - Passing primitive data type arguments
  - Passing reference data type arguments
- Arbitrary number of arguments
  - You can use a construct called **varargs** to pass an arbitrary number of values to a method
- Parameter names
  - A parameter can have the same name as one of the class' fields: to **shadow the field**

Constructors

- Constructor declaration
- Default constructor
- Constructor overloading
- Constructor chaining
- Private constructor
- How about destructors?

```
public class Account {
    private String name;
    private float balance;

    public Account() {
        name = "default";
        balance = 100;
    }

    public Account(String name) {
        this.name = name;
        balance = 100;
    }

    public Account(String name, float balance) {
        this.name = name;
        this.balance = balance;
    }
}
```

Creating and Using Objects

- Creating objects
  - Declaration, instantiation, initialization
  - The reference returned by the new operator does not have to be assigned to a variable
    - Example: `new Rectangle(100, 50).getArea()`

```
Account a = new Account("Garfield", 8);
```

- Referencing an object's fields
  - Use a simple name for a field within its own class
  - Code outside: `objectReference.fieldName`
- Calling an object's methods
- Garbage collector
  - An object is **eligible** for garbage collection when there is **no more reference** to it
  - The `finalize()` method
  - `System.gc()`

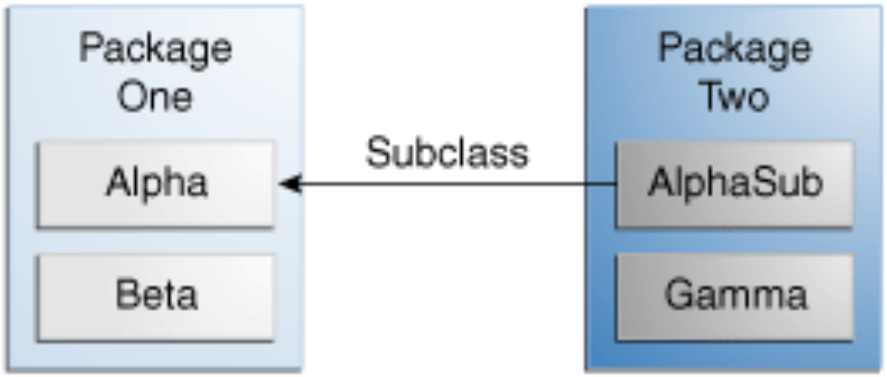
Access Control

- At the top level
  - public, or package-private (no explicit modifier)
- At the member level
  - public, private, protected, or package-private (no explicit modifier)

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Access Control (cont.)

- Example



- Visibility of members in the `Alpha` class

Modifier	`Alpha`	`Beta`	`AlphaSub`	`Gamma`
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Class Members (Static Members)

- Instance variables vs. class variables
- The *static* modifier
- Class variables are referenced by the class name itself
  - Static fields can also be referenced by an object reference, but discouraged
- Class methods should be invoked with the class name
  - Static methods can also be referenced by an object reference, but discouraged
- Combinations of instance and class variables/methods
- **Constants**: static modifier in combination with the final modifier
- Singleton

Initializing Fields

- You can often provide an initial value for a field in its declaration
- Instance variables can be initialized in constructors
- For class variables: **static initialization blocks**

```
static {
        // whatever code is needed for initialization goes here
}
```

- An alternative: private static method
  - Advantage: can be reused later if you need to reinitialize the class variable

Practice

- Design and implement a *Point* class
  - Fields (private): *xPos*, *yPos*
  - A set of constructors, including a default constructor
  - Getters and setters for *xPos* and *yPos*
- Design and implement a *Circle* class
  - Fields (private): *radius*, *center* (represented as *Point*)
  - A set of constructors, including a default constructor
  - Getters and setters for *radius* and *center*
  - *getArea*: compute and return the area of the circle
  - *toString*: return the information of the circle
- Design and implement three static methods
  - *Circle[] generate(int amount, float maxRadius, Point upperLeft, Point lowerRight)*
  - *Circle max(Circle... circles)*
  - *void sort(Circle[] circles)*

Nested Classes

- Java allows you to define a class within another class

```
class OuterClass {
        ...
        class NestedClass { ... }
}
```

- Nested classes are divided into two categories:
  static nested classes & inner classes
- A nested class is a member of its enclosing class
  - Can be declared private, public, protected, or package private
  - Inner classes have access to other members of the enclosing class, even if they are declared private
- Reasons for using nested classes
  - Logically grouping classes that are only used in one place
  - Increasing encapsulation, leading to more readable and maintainable code

Static Nested Classes

- A static nested class is behaviorally a **top-level class** that has been nested in another top-level class for packaging convenience
- Static nested classes are accessed using the enclosing class name
- A static nested class **cannot refer directly** to instance variables or methods

Inner Classes

- An inner class is **associated with an instance** of its enclosing class, and has **direct access** to that object's methods and fields
- It cannot define any static member itself
- To instantiate an inner class:

```
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```

- The shadowing issue: an example

Local Classes

- Classes defined in a block (a group of zero or more statements between balanced braces)
- Typically defined in the body of a method
- A local class has access to the members of its enclosing class
- A local class has access to local variables that are **declared final or effectively final**
- Declarations of a type in a local class **shadow** declarations in the enclosing scope

Enum Types

- An enum type: a special data type that enables for a variable to be a set of predefined constants
- Java enum types are **much more powerful** than their counterparts in other languages
  - All enums implicitly extend `java.lang.Enum`
  - The enum class body can include methods and other fields
  - The constructor for an enum type must be package-private or private access
  - Demo

Practice

- Implement a class named *Stack*, which stores *int* elements, and provides the following methods:
  - *void push(int x)*, *int pop()*, *int size()*, *void clear()*
- Implement an inner class *StackIterator*, which supports the iteration through stored values
  - *void reset()*, *boolean hasNext()*, *int next()*
- Experiment with the *Stack* and *StackIterator* classes in the *main* method