Overview on Java Programming

CSCI4180 tutorial 2

Qin Chuan

Outline

- Java program structure
- Java language basics
- Array
- OOP
  - Inheritance
  - Polymorphism
  - Abstract classes and interface

Array

- An array is a programming language construct (data structure)
  - hold and organize a fixed number of values of a single type
  - each item in an array is called an element
  - each element is accessed by its numerical index
  - index must be an integer and start from 0
- Declaration and array creation
  - type[] array_name = new type[length];
    - int[] i = new int[1000];
  - type[] array_name = {value1, value2, ...};
  - type may be a primitive type, class type or even an array type

Properties of Array

- A bounded (fixed length) and indexed collection of elements of the same type
- Array length is fixed at the time of creation
- Element access is done using an index inside [ ]
  - array_name[0]
- To get the length (size) of an array:
  - array_name.length

Array of Object References

- Creating a new array ≠ Creating members

```
Octopus[] deck;     // a null Octopus array reference
deck = new Octopus[10];  // initially, deck[0] = … = null
deck[0] = new Octopus();
deck[1] = deck[0];
deck[2] = new Octopus();
```

- Multidimensional arrays
  - elements of an array could be arrays
  - each dimension is an array of array references
  - e.g.: `double[][] number = new double[2][3];`

Outline

- Java program structure
- Java language basics
- Array
- OOP
  - Inheritance
  - Polymorphism
  - Abstract classes and interface

Inheritance

- Reuse and build related classes
- The subclass inherits the methods and data (characteristics, behaviors) defined by the superclass
- The subclass can also add new fields or methods, or modify (override) the inherited ones
- Proper inheritance creates an ***is-a*** relationship, meaning the child *is a* more specific version of the parent

Inheritance

- Syntax

```
class Subclass extends Superclass{
   ...
}
```

- Subclasses can be defined conveniently using inheritance
- Several subclasses can inherit from the same superclass
- But one subclass cannot inherit from multiple superclasses
- A subclass can re-define methods and fields
- We will talk this issue (override and hide) later
- A subclass can add new fields and/or methods as well

What Are Inherited?

- All public and protected members (field variables and methods) are inherited
- All private members (field variables and methods) are not inherited
- Modifiers public, private and protected DO NOT ONLY dictate what would be inherited, they ALSO affect the accessibility of a member

Privacy (Member Level)

- A member can be declared with the modifiers public, protected and private
  - public: a member is accessible anywhere
  - private: a member is ONLY accessible by that class
  - protected: a member is ONLY accessible by classes in the same package AND the subclasses of that class
  - no modifier (package private): a member is ONLY accessible by classes in the same package

Privacy (Class Level)

- A class can be declared with modifier public or no modifier
  - public: the class is visible to all classes
  - no modifier (package-private): the class is only visible within its own package

Super Construction

- On creating a new object, a constructor of the corresponding class will be invoked to initialize the object
  - Constructors are not inherited, although they are public
- To initialize inherited variables in subclass
  - call the parent's constructor super(…) to set up the "parent's part (fields)" of the object
  - it should be the first line to call super(…)
- The super reference can also be used to reference other variables and methods defined in the parent's class
  - super.methodName(parameters);

Class Members and Instance Members

- Class members belong to the class, rather than to an instance of the class
  - Defined with the keyword static
  - E.g.: the main method
  - Usually use the class name (not the object reference) to refer to the members
    - Math.sqrt(4); //square root
    - Math.PI; // get the value of Pi
- Instance members
  - Declared without keyword static

Class Members and Instance Members

- Access levels
  - Instance methods can access both instance members and class members directly
  - Class methods can access class members directly and **cannot** access instance variables or instance methods
    - they must use an object reference.
    - class methods cannot use the this keyword as there is no instance for this to refer to

Outline

- Java program structure
- Java language basics
- Array
- OOP
  - Inheritance
  - Polymorphism
  - Abstract classes and interface

Polymorphism

- Dynamic binding
- An object of the sub-class is also considered to be an object of the super-class (is-a relationship)
- By overriding a method declared in super-class, we may send the same message to different objects of different class types

Override

- An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass overrides the superclass's method
- The version of the overridden method that gets invoked is always the one in the subclass

Hide

- If a subclass defines a class method with the same signature as a class method in the superclass, the method in the subclass hides the one in the superclass
  - The version of the hidden method that gets invoked depends on the actual type of the object
- A field variable in the subclass that has the same name as a field in the superclass hides the superclass's field, even if their types are different
  - Not recommended
  - Within the subclass, the version of the hidden field is distinguished by using keywords this and super

Example: Override vs Hide

```
public class Animal {
     public static void testClassMethod() {
         System.out.println("The class method in Animal.");
     }
     public void testInstanceMethod() {
         System.out.println("The instance method in Animal.");
     }
}
```

```
public class Cat extends Animal{
    public static void testClassMethod() {
        System.out.println("The class method in Cat.");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Cat.");
    }
}
```

- Cat myCat = new Cat();
- myCat.testInstanceMethod(); //override
  - The instance method in Cat.
- ((Animal)myCat).testInstanceMethod(); //override
  - The instance method in Cat.
- myCat.testClassMethod(); //hide, the version in subclass in invoked
  - The class method in Cat.
- ((Animal)myCat).testClassMethod(); //hide, the version in superclass in invoked
  - The class method in Animal.

Keyword final

- Declare a class final
  - prevents the class from being subclassed
- Declare a method final
  - the method cannot be overridden by subclasses

Outline

- Java program structure
- Java language basics
- Array
- OOP
  - Inheritance
  - Polymorphism
  - Abstract classes and interface

Abstract Class

- Abstract Methods
  - An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon)
  - abstract <return-type> methodName (arguments);
- Abstract Class
  - It may or may not include abstract methods
  - cannot be instantiated
  - can be subclassed

Interface

- An interface is
  - a specification
  - a collection of **abstract** methods and constants
  - **Do not** need the modifier **abstract** like in an abstract class, because all methods in an interface are abstract
- Any class that implements an interface is guaranteed to provide all methods defined in the interface

Interface Definition

- All the methods in an interface are abstract (none is given a body), and there is no modifier abstract

```
public interface Doable
{
   public void doThis();
   public int doThat();
   public void doThis2 (double value, char ch);
   public boolean doTheOther (int num);
}
```

- Methods are public by default: the modifier can be omitted

Implements An Interface

- All methods defined in the interface must be implemented

```
public class CanDo implements Doable
{
    public void doThis (){
        // must be implemented
    }
    public void doThat (){
        // must be implemented
    }
    public void doThis2 (double value, char ch){
        // must be implemented
    }
    public boolean doTheOther (int num){
        // must be implemented
    }
}
```

Abstract and Interface

- Definition
  - Abstract class may or may not contain abstract methods
  - Interface only contains abstract methods
- Usage
  - public class Class_Name extends Abstract_Class_Name{ }
  - public class Class_Name implements Interface_Name{ }
- A class can only extends one abstract class, but can implements multiple interfaces

References

- Java language specifications
  - http://java.sun.com/docs/books/jls/
- Java tutorial
  - http://docs.oracle.com/javase/tutorial/index.html
- Java API
  - http://docs.oracle.com/javase/6/docs/api/

Thanks!