Classes/Objects/Interfaces (Savitch, Various Chapters)

TOPICS

• Classes
• Public versus Private
• Static Data
• Static Methods
• Interfaces

Classes

• Classes are the basis of object-oriented (OO) programming.
• They encapsulate functionality to form powerful abstractions of real world objects.
• What can classes be used for? Classes have many different uses, for example:
  ▪ Data Structures
  ▪ Code Libraries
  ▪ Java Programs
  ▪ Complex Objects

Classes as Data Structures

• Just like a struct in C and C++ (no code), for example:
  public class Student {
      public String firstName;
      public String lastName;
      public Date birthDate;
      public Address homeAddress;
      public double gradePointAverage;
  }

Classes as Code Libraries

• Just like a library in a procedural language (no data) like C or C++, for example:
  public class Math {
      public static final double PI = 3.14159;
      public static double sin(double a) {
          ...
      }
      public static double exp(double a) {
          ...
      }
      public static double log(double a) {
          ...
      }
      public static double sqrt(double a) {
          ...
      }
  }
Classes as Small Programs

• Just like a program in a procedural language like C or C++, for example:

```java
class MySmallProgram{
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

Classes as Large Programs

• Just like a program in a procedural language like C or C++, for example:

```java
class MyLargeProgram{
    // lots of data
    public static void main(String args[]) {
        // lots of code
    }
    // lots of methods
}
```

Classes as Complex Objects

• No comparable example in a procedural language like C or Pascal!

```java
class MyClass {
    // class variables (static)
    // instance variables (non-static)
    // no main method
    // class methods (static)
    // instance methods (non-static)
}
```

Using Different Class Types

// Data Structure
Student students[] = new Student[100];
students[0].firstName = "Christopher";

// Code Library
System.out.println(Math.sin(1.0));

// Java Programs
$ java MySmallProgram

// Complex Objects
MyClass myClass = new MyClass();
myClass.initialize();
Public versus Private

- **public**
  - Can access the class, method, or data by name outside defining class
- **private**
  - Can access the class, method, or data by name only inside defining class
- Classes generally specified as **public**
- Instance variables usually are **private**
- Methods can be **public** or **private**

Static (versus non-Static)

- Local data resides in a method, class and instance data resides in the class.
- Data defined in the class can be of two types:
  - Data may belong to the class (only one copy that has the same value for all objects of that class)
  - Data may belong to the object (separate storage and different values for each instance)
- Class data is identified by **static**
- Instance data is not be **static**

Class versus Instance: Data

```java
public class MyClass {
    static int int0;
    double real0;
}
```

- `int0` resides in the class (1 copy), use class name to access: `MyClass.int0 = 123;`
- `real0` resides in an instance (many copies):
  ```
  MyClass m = new MyClass;
  m.real0 = 1.234;
  ```

Class Methods

- Notation: **class.method()**
- Does not require instantiation, should be called using the class name.
- Class name is not necessary if within the class itself.
- Marked as static, and cannot access instance data, since no instance is used to call them.
- Useful for library methods that do not need any associated data: for example `Math.sin();`
Instance Methods

- Notation: `objectname.method()`
- Must be called on an object instantiated from a class.
- Most objects are instantiated with the new keyword: `String word = new String("Whatever");`
- For example, calling `word.length()` requires an object of type `String` called `word`.
- The `length()` method uses the data for the specific instance it is called on.

Constructors

- Q: How to initialize instance data when creating an object? A: Use a constructor.
- Constructors are used to create objects and send them data.
- Always named after the class, arbitrary parameters, but no return value!
- Example:
  ```java
  Scanner s = new Scanner(System.in);
  ```

Methods inside a class

- Order of writing methods is arbitrary
  - Generally constructors are written first.
- What if two methods need to share data?
  - Can pass data around as parameters and return values.
  - Alternative: specifying class or instance data allows access from multiple methods.

Encapsulation

- Consider example of driving a car
  - We see and use break pedal, accelerator pedal, steering wheel – we know what they do.
  - We do not see mechanical details of how they do their jobs.
- We can divide class definition into different parts to support encapsulation:
  - Class interface
  - Class implementation
### Interface/Implementation

- **A class interface**
  - Tells the user or programmer what the class does
  - Gives headings for public methods and comments about them
- **A class implementation**
  - Contains private variables
  - Includes definitions of public and private methods, i.e. how the class does what it does

### Interfaces

- Interface definition is pure functionality, a set of methods and a description of what they do (Javadoc).
- Instance variables in the class usually are declared as private to prevent direct access.
- Provide public accessor (getter) methods to read data.
- Provide public mutator (setter) methods to write data.
- Add comments before each public method heading that fully specifies how to use method.
- Hide everything else: data structures, helper methods, algorithms - all implementation details.

### Interface Example

```java
public interface Trigonometry {
    public double sin(double angle);
    public double cos(double angle);
    public double tan(double angle);
}

public class Math implements Trigonometry {
    public double sin(double angle) { ... code ... }
    public double cos(double angle) { ... code ... }
    public double tan(double angle) { ... code ... }
}

Trigonometry trig = new Math();
```

### Encapsulation/Interfaces

A well encapsulated class definition:
Why Interfaces?

- Interfaces are pure functionality, a designer can specify what the software does without worrying about how.
- Provides a methodology for partitioning large systems without having to consider all of the details.
- Interfaces can (and should) be designed based on user requirements and the intended functionality.
- Interfaces hide implementation details, user of class does not need to see internal code or data structures.
- Allows shipment of class files to protect intellectual property while still communicating functionality.