INFO 5100

Application Engineering Design

Java Classes and Objects

Daniel Peters

d.peters@neu.edu

- Lecture
  1. Java Classes and Objects
     1. Java Intro
     2. Java Object Oriented Programming
     3. Package organization of java classes
     4. Java Class Details

Driver Class specification

```
public class Driver {


   // main() method
   public static void main(String [] args) {
         System.out.println("Hello World!");
         System.out.println("Bye World!");
   }
}
```

Java OOP: Class Specification

- Class specification
  - Each Java class is written (coded in java) in a single '***.java***' text file
  - Java code (classes) **MUST** be compiled using the java compiler:

javac *Driver*.**java**

Java OOP: Object Instantiation

- Object Instantiation
  - A compiled Java program (.class) is executed (begins running) using the **'java'** command:

    java *Driver*

    •program execution ALWAYS begins in **main()** method
  - A running java program executes java statements one after another beginning with the first java statement in (the designated) **main()** method

Java OOP: Object Instantiation

- Classes are instantiated into useable objects
  - Program execution begins in **main()** method in the Driver class
    - Java program executes **java statements** one after another beginning in the **main()** method.
    - ALL java statements **end with a semicolon ';'**
  - Java code in **main()** method will:
    - Instantiate objects from class specifications
    - Use objects for ALL program execution
  - Program execution ends when **main()** method exits

Java OOP: Object Instantiation

- Program execution uses program variables
- Program variables are named memory locations used to contain data for program execution
- Java is a Statically Typed language
  - All program variables MUST BE DECLARED (both **Type** and **name** announced to java compiler) before they can used by java program

Java Object Oriented Programming

- Object Oriented Programming (OOP)
  - Fundamental to Java language
  - Everything in Java is related to a Class
  - Required by Java (not optional like C++)

Java Object Oriented Programming

- Primary Object Oriented Programming Concepts
  1. Abstraction
  2. Encapsulation
  3. Inheritance
  4. Polymorphism

OOP Concepts: Abstraction

- Abstraction
  - Data Hiding with access modifiers
  - Functionality Hiding with API
  - Supports SOLID design principles
  - Black Box
    - Restricting visibility to inner details
  - Provides design simplification
  - Allows Focus on "**What**" by hiding "**HOW**"

OOP Concepts: Encapsulation

- Encapsulation
  - Data and methods which operate on that data contained in the same class.
  - Supports SOLID design principles
  - Facilitates design, development and maintenance of software
  -

OOP Concepts: Inheritance

- Inheritance
  - Allows a child subclass to inherit from a parent subclass.
  - Supports SOLID design principles
  - Provides design organization and simplification
  - Promotes code reuse

OOP Concepts: Polymorphism

- Polymorphism
  - Existing in many forms
  - Supports SOLID design principles
  - Provides design simplification
  - Provides for design flexibility and extensibility

Java Object Oriented Programming

- Classes are specified
- Classes are instantiated into useable objects
- Objects are used as the functional building blocks of the executing java program

Person1 Class

```
public class Person1 {
        public static String name;          // class data
    }
```

NOTES:

1. '**public**' class data violates encapsulation and serves only for a trivial example.
2. Java keyword 'static' specifies 'name' is part of class Person1

Driver Class

```
public class Driver {
   // main() method
   public static void main(String [] args) {
         Person1.name = "Dan";
         System.out.println(Person1.name);
         Person1.name = "Jim";
         System.out.println(Person1.name);
   }
}
```

CONSOLE OUTPUT

Dan

Jim

Person1 Observations

1. Single global Person1 class must share its data (name)
    1. Cannot model more than one person well
2. Shared data restricts multithreaded programming
    1. Multithreaded programming leverages multi-core devices (servers, laptops, tablets, smartphones) for increased performance and scalability

Person2 Class

```
public class Person2 {
        public String name;          // object data
    }
```

NOTES:

1. '**public**' class data violates encapsulation and serves only for a trivial example.
2. Non-static 'name' is part of each object instantiated from Person2 class

Driver Class

```
public class Driver {
   public static void main(String [] args) {
    Person2 objectDan = new Person2();
    objectDan.name = "Dan";
    System.out.println(objectDan.name);
    Person2 objectJim = new Person2();
    objectJim.name = "Jim";
    System.out.println(objectJim.name);
   }
}
```

Java OOP: Object Instantiation

- The **main()** method in class **Driver**:

```
public class Driver {
   public static void main(String [] args) {
         Person2 objectDan = new Person2();
    . . .
   }
}
```

Java OOP: Object Instantiation

- Instantiate (Create) an Object from Class specification in **three** steps
  1. **Declare:** declare program reference variable
  2. **Instantiate:** create Person2 object from class using keyword '*new*' AND Person2 class constructor
  3. **Assign:** (write/save) Instantiated Person2 object reference to (memory location named) 'objectDan'

Java OOP: Object Instantiation

- **Declare**: declare program reference variable **‘objectDan’**:

  Person2 objectDan;

  – Type: class **Person2**

  – Name: **objectDan**

Java OOP: Object Instantiation

- **Instantiate:** create Person2 object from class using keyword '***new***' AND Person2 default class constructor

  ***new* Person2();**

Java OOP: Object Instantiation

- **Assign:** (write/save) Instantiated **Person2** object to (memory location named) **‘objectDan’**
- ALL DONE IN A SINGLE Java STATEMENT
  - **Person2 objectDan** = *new* **Person2();**

CONSOLE OUTPUT

Dan

Jim

Java OOP: Object Usage

- Objects are used as building blocks for a java program
- With one exception, we must instantiate java class as an object to use its members.
  - A class' static members may be used without instantiation as an object.
- Use the '**.**' (dot) to access public data and methods in static classes and instantiated objects

Person2 Observations

1. Multiple objects can be instantiated (created) from Single global Person2 class.
2. Each Person2 object instance can model a person well.
3. Multiple Person2 objects model multiple persons.
4. Object instance data supports multithreaded programming on multi-core hardware.

Person3 Class

```
public class Person3 {
        private String name;        // object data
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
    }
```

- NOTE: Person3 supports encapsulation.

Driver Class

```
public class Driver {
   public static void main(String [] args) {
    Person3 objectDan = new Person3();
    objectDan.setName("Dan");
    System.out.println(objectDan.getName());
    Person3 objectJim = new Person3();
    objectJim.setName("Jim");
    System.out.println(objectJim.getName());
   }
}
```

CONSOLE OUTPUT

Dan

Jim

Person3 Observations

1. Encapsulation restricts data access to Person3 data by Person3 methods ONLY
    1. Supports SOLID design principles
    2. Simplifies Design
    3. Increases maintainability
    4. Facilitates extensibility
2. Person 3 class has all the benefits of Person 2 class

Person4 Class

```
public class Person4 {
        private String name; // object data
        public Person4() { // default constructor
                super();
                this.name = "Joe";
         }
        public Person4(String name) {
                super();
                this.name = name;
        }
        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
        @Override
        public String toString() { return this.name; }
}
```

Driver Class specification

```
public class Driver {
    public static void main(String [] args) {
        Person4 object = new Person4();
        System.out.println(object);
        Person4 objectDan = new Person4("Dan");
        System.out.println(objectDan.getName());
        Person4 objectJim = new Person4();
        objectJim.setName("Jim");
        System.out.println(objectJim.getName());
    }
}
```

CONSOLE OUTPUT

Joe

Dan

Jim

Person4 Observations

1. Java compiler provides a trivial (initialize object state to zero) default constructor for instantiating an object from a class.
    1. Implicit default constructor provided ONLY when NO CONSTRUCTORS are specified.
2. Providing explicit class constructors allows for custom initialization of the state (i.e. data) of each instantiated object.
3. Person 4 class has all the benefits of Person 3 class

Java Package

- Package
  - Organization of Java class libraries
    - Namespace organization
    - File system organization
      - Source files (.java)
      - Class files (.class)
  - Class libraries in a package are related
  - Hierarchical dot'.' separated name

Java Package

- Package Name Convention
  - All lower case package name begin with top level domain
    - edu, com, org, mil, ca, de, uk
  - Followed by organization name
    - ibm, neu, mit, microsoft
  - Followed by any groups, projects or sub-projects within the organization

Java Package

- Package name examples
  - java.lang
  - java.util
  - java.awt
  - java.swing
  - edu.neu.csye6200.lecture1.misc

Class

- Class

  public class MyName [*extends* MySuperClass] [*implements* MyInterface] {

  - Data
  - Constructor (a special method)
  - Method
  - }

- All outer class definitions MUST BE public
  - Inner class (defined in a class) may be private

Class (cont'd)

- Class
  - Concrete
    - Declared
    - Fully implemented methods
    - Can be instantiated to create objects

Class (cont'd)

- Class
  - Abstract
    - Declared: public, private or protected members
    - Partially implemented
    - Contains one or more abstract methods
    - Data
      - final or non final
      - static or non-static
    - Must be extended (keyword extends)
    - Cannot be instantiated (without completing implementation)

Class (cont'd)

- Class
  - Interface
    - Contains ONLY (implicitly) public methods
      - Implicitly abstract methods are unimplemented
      - Java 8 'default' Methods are implemented
      - Java 8 'static' Methods are implemented
    - Data
      - static (class variables) ONLY
      - final (immutable constant values) ONLY
    - Must be implemented (keyword implements)
    - Cannot be instantiated (without completing implementation)

Class (cont'd)

- Class
  - Data
    - Attribute
    - Field
  - Constructor
    - One ore more *special* Method to instantiate and initialize objects
  - Method
    - Function
    - Operation
    - Behavior

Class (cont’d)

- Class
  - Data declaration
    - [static] [final] [public | protected | private] type name
    - [= initializer] ;

Data Types

- Primitive types
  - byte, short, int, long, float, double, boolean, char
  - Passed by value
  - Typically, Stack memory allocation

  https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Data Types

- Reference types
  - Class
  - Passed by reference
  - Heap memory allocation
    - Automatic Garbage Collection (GC)
    - NEVER TO FREE (C++ delete) heap allocation

Class (cont'd)

- Class
  - Static: class global data
    - Single instance of data
    - Associated with class
    - Object instantiation not required
    - Program scope
  - Static: method, program scope
    - Can not be overridden
    - Should always be accessed in 'static' way, i.e., with class name and not with an object reference
      - System.**out**.println() (***out*** is a *static* member of System)

Class (cont'd)

- The following are NEVER '**static**' in Java
  - Constructor
    - Allowed in other Object-Oriented languages to initialize static data members
      - Java instead uses static initialization block

```
public class Driver {
        static {
        …
        }
}
```

  - Outer class
    - Inner class may be a '**static**' member of outer class

Class (cont'd)

- Class
  - Non-static: object instance data
    - Default
    - Independent instance with each object created
    - Object instantiation required
      - Heap memory allocation
    - Object Reference assigned to variable
    - Reference points to Object in heap memory

Class (cont'd)

- Class
  - Non-static: object instance method
    - Default
    - Object instantiation (*new*) IS required
      - Does not exist until object is created
    - Object Reference assigned to variable
      - Method is called using object reference
        - » double price = new Item().getPrice();
      - '**this**' in object instance method is reference to current object on heap
    - Reference points to Object in heap memory

Class (cont'd)

- Class
  - final
    - Immutable data item: Constant data (init once)
      - Independent constant with each object
        - » final int JOB_ID = 347;
        - » Final String LABEL = "EMPLOYEE";
      - Single instance of data
        - » static final int ERROR_CODE = 147;
        - » static final String ERROR= "Invalid Input Parameter"
    - Immutable method
      - Cannot be overridden by inheritance (like C++ non-virtual method)
      - NEVER use '**final**' for a class constructor

Class (cont'd)

- Class
  - Access Modifiers
    - Public
    - Protected
    - (Default)
    - Private
  - Provides Data Hiding (Abstraction)
    - Applicable individually to each member of class

Class (cont'd)

- Class
  - Access Modifiers
    - Public
      - All access
        - » Accessible by classes within package
        - » Accessible by sub-class
        - » Accessible by classes outside package

Class (cont'd)

- Class
  - Access Modifiers
    - Protected
      - Class, Package **and Sub-class** access
        - » Accessible by classes within package
        - » **Accessible by sub-class**
        - » NOT Accessible by classes outside package

Class (cont'd)

- Class
  - Access Modifiers
    - Default: Neither Public, Protected nor Private
      - Package Private
        - » Accessible by classes within package
        - » **NOT Accessible by sub-class**
        - » NOT Accessible by classes outside package

Class (cont'd)

- Class
  - Access Modifiers
    - Private
      - Class private
        - » **ONLY Class itself has access to private members**
        - » NOT Accessible by classes within package
        - » NOT Accessible by sub-class
        - » NOT Accessible by classes outside package

Class (cont'd)

- Class
  - Constructor
    - *Special* Method used to instantiate objects
    - Constructor NAME is IDENTICAL to class name
    - MUST NOT specify a return value type OR void
    - Default Constructor
      - No arguments
      - Compiler provided IF NO CONSTRUCTORS
    - Multiple Constructors
      - Overloaded
        - » Different signatures (i.e., number and types of args)
      - Provides **Static Polymorphism**

Class (cont'd)

- Class
  - Constructor
    - Never '**static**' in Java
      - Allowed in **C#**, a ***static*** *constructor* would initialize ***static*** data members: Java uses static initialization block instead
    - Never '**final**'
    - **Only Allowable Modifiers for constructor:**
      - **public**
      - **private**
      - **protected**

Class (cont'd)

- Class
  - Method
    - Also called function, operations, behaviors
    - Abstract: declaration only: no implementation
    - Concrete: declaration and implementation
    - MUST specify a return value type OR **void**
    - Overloaded Methods
      - Same names
      - Different signatures (i.e., number and types of arguments)
      - Different return types DOES NOT distinguish methods
      - Provides **Static Polymorphism**
    - Override (@Override) **run-time Polymorphism**

Java Class Summary

- Class Summary
  - Single Outer class in **.java** source code file
    - Name of class is ALSO used for **.java** file name
  - Outer class MUST ALWAYS '**public**'
  - Outer class MUST NEVER BE '**static**'
  - Inner class is a data member of outer class
    - Allowed to be '**private**', '**protected** or '**public**'
    - Allowed to be '**static**'

Java Class Summary

- Class Summary
  - Concrete
    - Fully implemented methods
  - Abstract
    - Contains one or more abstract methods
    - *CANNOT be instantiated - MUST be extended
- Interface
    - Contains public abstract, default and static methods
    - *CANNOT be instantiated - MUST be implemented
- * NOTE: UNLESS implementation is completed

Benefits

- Java class Benefits
  - Encapsulation
    - Data and Method associated together in class
  - Abstraction
    - Data hiding
      - Access Modifiers
        - » Public, Private, Protected
    - Functionality hiding
      - Abstract method as API
      - Interface as API