CMSC 132:
Object-Oriented Programming II

Java Support for OOP

Department of Computer Science
University of Maryland, College Park
"this" Reference

Description

- Reserved keyword
- Refers to object through which method was invoked
- Allows object to refer to itself
- Use to refer to instance variables of object
“this” Reference – Example

class Node {
  value val1;
  value val2;
  void foo(value val2) {
    ... = val1;       // same as this.val1 (implicit this)
    ... = val2;       // parameter to method
    ... = this.val2;  // instance variable for object
    bar( this );      // passes reference to object
  }
}

Also used in constructors to invoke another constructor in the same class.
Inheritance

Definition
- Relationship between classes when state and behavior of one class is a subset of another class

Terminology
- Superclass / parent → More general class
- Subclass → More specialized class

Forms a class hierarchy
- Helps promote code reuse
“super” Reference

Description

- Reserved keyword
- Refers to superclass
- Allows object to refer to methods / variables in superclass

Examples

- super.x // accesses variable x in superclass
- super() // invokes constructor in superclass
- super.foo() // invokes method foo() in superclass
References & Aliases

Reference
- A way to get to an object, not the object itself
- All variables in Java are **references** to objects

Alias
- Multiple references to same object
- “x == y” operator tests for alias
- x.equals(y) tests contents of object (potentially)
Implementing Equals

**Approach we want to use (assuming class A)**

```java
public boolean equals(Object obj) {
    if (obj == this)
        return true;
    if (!(obj instanceof A))
        return false;
    A a = (A)obj;
    /* Specific comparison based on A fields appears here */
    return true;  // or whatever
}
```

**Example:** See equalsMethod package
Constructor

**Description**

- Method invoked when object is instantiated
- Helps initialize object
- Method with same name as class \textit{w/o} return type
- Default parameterless constructor
  - If no other constructor specified
  - Initializes all fields to 0 or null
- Implicitly invokes constructor for superclass
  - If not explicitly included
class Foo {
    Foo( ) { … }       // constructor for Foo
}

class Bar extends Foo {
    Bar( ) {          // constructor for Bar
        // implicitly invokes Foo( ) here
        …
    }
}

class Bar2 extends Foo {
    Bar2( ) {            // constructor for Bar
        super();        // explicitly invokes Foo( ) here
        …
    }
}
Three Levels of Copying Objects

1. Reference copy
   - Makes copy of reference
   - \( x = y; \)

2. Shallow copy
   - Makes copy of object
   - \( x = y.clone(); \)

3. Deep copy
   - Makes copy of object \( z \) and all objects (directly or indirectly) referred to by \( z \)
Cloning

- Creates identical copy of object using clone() method

Cloneable interface

- Supports clone() method
- Returns copy of object
  - Object class version makes a shallow copy
  - Over-ride it if you implement Cloneable

EXAMPLE: Cloning
Initialization Block

Definition

- Block of code used to initialize static & instance variables for class

Motivation

- Enable complex initializations for static variables
  - Control flow
  - Exceptions
- Share code between multiple constructors for same class
**Initialization Block Types**

- **Static initialization block**
  - Code executed when class loaded

- **Initialization block**
  - Code executed when each object created
    (at beginning of call to constructor)

**Example**

```java
class Foo {
    static {
        A = 1;
    } // static initialization block

    {
        A = 2;
    } // initialization block
}
```
Variable Initialization

- Variables may be initialized
  - At time of declaration
  - In initialization block
  - In constructor

Order of initialization

1. Declaration, initialization block
   (in the same order as in the class definition)
2. Constructor

EXAMPLE: Octopus.java
Garbage Collection

Concepts

- All interactions with objects occur through reference variables
- If no reference to object exists, object becomes garbage (useless, no longer affects program)

Garbage collection

- Reclaiming memory used by unreferenced objects
- Periodically performed by Java
- Not guaranteed to occur
- Only needed if running low on memory
Destructor

Description

- Method with name `finalize()`
- Returns `void`
- Contains action performed when object is freed
- Invoked automatically by garbage collector
  - Not invoked if garbage collection does not occur

Example

```java
class Foo {
    void finalize() { … }    // destructor for foo
}
```
Method Overloading

Description
- Same name refers to multiple methods

Sources of overloading
- Multiple methods with different parameters
  - Constructors frequently overloaded
  - Redefine method in subclass

Example

```java
class Foo {
    Foo() { ... } // 1st constructor for Foo
    Foo(int n) { ... } // 2nd constructor for Foo
}
```
Package

Definition
- Group related classes under one name
- Helps manage software complexity
  - Separate namespace for each package
    - Package name added in front of actual name
  - Put generic / utility classes in packages
    - Avoid code duplication

Example

    package edu.umd.cs;    // name of package
Package – Import

- **Import**
  - Make classes from package available for use

- **Example**

```java
import java.util.Random; // import single class
import java.util.*;     // all classes in package
...
```

```
// class definitions
```

Alternatively, use fully qualified name everywhere:

```java
java.util.Random r = new java.util.Random();
```
Scope

- Part of program where a variable may be referenced
- Determined by location of variable declaration
  - Boundary usually demarcated by {  }

Example

```java
public MyMethod1() {
    int myVar;
    ...
}
```

myVar accessible in method between {  }
package edu.umd.cs;
public class MyClass1 {
    public void MyMethod1() {
        ...
    }
    public void MyMethod2() {
        ...
    }
}
public class MyClass2 {
}

Scope – Example

Example

Scopes

Package

Class

Method
Modifier

Description

- Java keyword (added to definition)
- Specifies characteristics of a language construct

(Partial) list of modifiers

- Public / private / protected
- Static
- Final
- Abstract
Modifier

Examples

```java
public class Foo {
    private static int count;
    private final int increment = 5;
    protected void finalize { ... }
}

public abstract class Bar {
    abstract int go( ) { ... }
}
```
Visibility Modifier

Properties
- Controls access to class members
- Applied to instance variables & methods

Four types of access in Java
- Public
- Protected
- Package
- Private

Most visible
Least visible
### Visibility Modifier – Where Visible

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Visibility Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;public&quot;</td>
<td>Referenced anywhere (i.e., outside package)</td>
</tr>
<tr>
<td>&quot;protected&quot;</td>
<td>Referenced within package, or by subclasses outside package</td>
</tr>
<tr>
<td>None specified (package)</td>
<td>Referenced only within package</td>
</tr>
<tr>
<td>&quot;private&quot;</td>
<td>Referenced only <strong>within</strong> class definition</td>
</tr>
<tr>
<td></td>
<td>Applicable to class fields &amp; methods</td>
</tr>
</tbody>
</table>
Visibility Modifier

For instance variables
- Should usually be **private** to enforce encapsulation
- Sometimes may be **protected** for subclass access

For methods
- **Public methods** – provide services to clients
- **Private methods** – provide support other methods
- **Protected methods** – provide support for subclass
Visibility Modifier

```
package fooBar;
public class A {
    public int vPub;
    protected int vProt;
    int vPack;
    private int vPriv;
}

package fooBar;
public class B {
    can access vPub;
    can access vProt;
    can access vPack;
    cannot access vPriv;
}

package fooBar;
public class D extends A {
    can access vPub;
    can access vProt;
    cannot access vPack;
    cannot access vPriv;
}

package fooBar;
public class E {
    can access vPub;
    cannot access vProt;
    cannot access vPack;
    cannot access vPriv;
}

"Access" means access by name, e.g.:
a = new A();
a.vProt = 2;
```
Modifier – Static

Static variable
- Single copy for class
- Shared among all objects of class

Static method
- Can be invoked through class name
- Does not need to be invoked through object
- Can be used even if no objects of class exist
- Can not reference instance variables
Modifier – Final

Final variable
- Value can not be changed
- Must be initialized in every constructor
- Attempts to modify final are caught at compile time
- With reference variable, does NOT make object immutable

Final static variable
- Used for constants
- Example
  
  ```java
  final static int Increment = 5;
  ```
Modifier – Final

Final method
- Method *can not be overridden* by subclass
- Private methods are implicitly final

Final class
- Class *can not be extended*
- Methods in final class are implicitly final
- Example – class *String* is final