Static Members, Enumerations and Packages

Lecture 5

Example Class Declaration

```java
public class Pencil {
    private int defaultLength = 10;
    private String color;
    private int length;

    public Pencil (int length) {
        if (length > 0) {
            this.length = length;
        } else {
            this.length = defaultLength;
        }
    }

    public int sharpen (int amount) { . . . }
    public String toString () { . . . }
}
```

One Pencil Instance

```
defaultLength: 10
    color: red
    length: 6
Pencil()
    sharpen()
    toString()
```

Multiple Pencil Instances

```
defaultLength: 10
    color: black
    length: 6
Pencil()
    sharpen()
    toString()

defaultLength: 10
    color: red
    length: 3
Pencil()
    sharpen()
    toString()

defaultLength: 10
    color: blue
    length: 3
Pencil()
    sharpen()
    toString()
```

Object vs Class Members

- Class member: only one copy, which is shared by all instances
  - Keyword: static
    ```java
    static int defaultLength;
    static void reset() { . . . }
    ```

Multiple Pencil Instances

```
defaultLength: 10
    color: black
    length: 6
Pencil()
    sharpen()
    toString()

defaultLength: 10
    color: red
    length: 3
Pencil()
    sharpen()
    toString()

defaultLength: 10
    color: blue
    length: 3
Pencil()
    sharpen()
    toString()
```
aka Instance vs Static Members

- Static members available even before instances (objects) are created!
  - From outside of class: `classname.member`
  - From inside class: `classname` is optional
- Conversely, static members can *not* access instance members
  - `ie` this reference cannot be used
  ```java
  public static void reset() {
    length = defaultLength;
  }
  ```

Good Practice: Static Members

- Do *not* access static members through object references
- Use class names instead:
  ```java
  int t = Pencil.defaultLength;
  ```

Example: println

- System.out.println("Hello");
- What is System?
  - A class from Java library
    - See API documentation: `java.lang.System`
- What is `out`?
  - A static field of System (available from class)
    - Type: reference to an instance of PrintStream
- What is `println`?
  - An overloaded method in PrintStream
    - Different versions for printing String, int, boolean...

Example: main()

- ```java
  public class HelloWorldApp {
    public static void main(String[] args) {
      ...
    }
  }
  ```

Example

- See Artifact.java
  - Static members
    - Fields for: class creation time, first instantiation, most recent instantiation, total number of instantiations
    - Method for getting number of instantiations
  - Instance members
    - Field holding a float
    - Method for getting information (toString)
    - Constructor
    - Static initialization block (more on that later)
- See ArtifactTester.java
  - Note output showing different times

Constant Fields: final

- ```java
  final int[] DEFAULT_LENGTH = {10};
  ```
- Modifier `final` on field means it cannot change
  - For primitive type, effectively a constant
    ```java
    final int i1 = 53;
    final int i2 = (int) (Math.random()*20);  // i2++ must initialize
    ```
  - For objects, only the reference is constant
    ```java
    final Pencil p = new Pencil("blue");
    ```

- Often used in conjunction with static
  - Class-wide constant value
    ```java
    static final int DEFAULT_LENGTH = 10;
    ```

Good Practice: No Magic Numbers

- "Magic Number": a numeric constant in code
  for (int i=0; i < 365; i++) { ... }
- Some literals are acceptable
  - Booleans and references (true, false, null)
  - Integers: -1, 0, 1, 2
- The rest should all be avoided
  for (int i=0; i < DAYS_PER_YEAR; i++) { ... }
- See Java libraries (API, constant-values):
  - Integer.MAX_VALUE, Math.PI
  - Float.POSITIVE_INFINITY, Thread.MAX_PRIORITY
- Important benefits:
  - Single point of control over change
  - Legibility

Outdated (bad) Idiom: int enums

- Enumeration type: legal values a finite set of constants
  - Card suits (clubs, diamonds, hearts, spades)
  - Days of the week (D, M, T, W, R, F, S)
- This could be done with static final fields
  public class PlayingCard {
    public static final int CLUBS = 0;
    public static final int DIAMONDS = 1;
    public static final int HEARTS = 2;
    public static final int SPADES = 3;
    ...
  }
- Later, use these named constants
  if (trump == PlayingCard.CLUBS) { . . . }
- Problem: no type safety!
  if (trump == 23) { . . . }

Enum Types

- Declared like a class, keyword enum
  enum Suit {
    CLUBS, DIAMONDS, HEARTS, SPADES
  }
- These constants are (implicitly) static fields
  - Suit.trump = Suit.SPADES; //do not use new()!
  - if (trump == Suit.CLUBS) { . . . }
- Can also contain fields & methods (and nested types)
- Automatically provided (static) methods include:
  - values() - returns array of constants
  - valueOf(String) - returns constant with that name
  - ordinal() - returns constant's position in declaration list
    Suit.CLUBS.ordinal() == 0;

Packages: Component Catalogs

- A package is a grouping of classes
  - Hierarchical: subpackages within packages
    - java.lang, java.util, java.util.logging
    - see http://java.sun.com/javase/6/docs/api
  - A package provides
    - Logical structuring: related classes are bundled
    - Encapsulation: another level of access control
    - Distinct namespace: classes in different packages can have the same name without conflict
    - Convention to guarantee uniqueness of package name: reverse of company's domain name
      - org.w3c.dom, edu.ohio-state.cse

Declaration

- Use package statement at top of source file
  - Must appear first, before any class declarations
  - public class Pencil { . . . }
- This file must be in a directory matching package name
  - Pencil.java in ???/edu/ohio-state/cse
- Eclipse handles this correspondence for you
- At most one package declaration in a file
- If there is no package declaration, class is in unnamed default package
  - This is fine only for very small programs (like the ones you will write for this class)

Access Control

- Another level of visibility: package
  - Default for members (public/private omitted)
  - Package-visible members are accessible by all classes in the same package
  - package edu.ohio-state.edu;
  - public class Pencil {
      private String color;
      int length;
      . . .
    }
- Classes are public or package (default)
  - Public classes available outside package
    - public class Math { . . . }
  - Package classes available only within same package
    - class Pencil { . . . }
Type Imports

- Fully-qualified type name is `package.class`
  
  ```java
type Imports

- Fully-qualified type name is package.class
  - Do not confuse this "." with member access
- Shorthand: import statement at top of file
  - To import a single `public` type
    ```java
    java.util.Date d = new java.util.Date();
    ```
  - To import all `public` types, use wildcard `*`
    ```java
    import java.util.*;
    ```
- * does not import subpackages
- All classes implicitly import `java.lang.*`
- Static members can be explicitly imported
  ```java
  import static java.lang.Math.exp;
  ```
  ```java
  exp(x); //instead of Math.exp(x)
  ```
- Can use wildcard `*` as well

Good Practice: Naming Conventions

- Avoid name conflicts with packages and reserved keywords
- Package names: lowercase letters
  ```java
  java.util, java.net, java.io, ...
  ```
- Class names: start with uppercase letter
  ```java
  Math, Pencil, PriorityQueue, ...
  ```
- Variable, field and method names: start with lowercase letters
  ```java
  x, out, myColor, abs(), getName(), isEven() ...
  ```
- Constant names: all uppercase letters
  ```java
  PI, DEFAULT_LENGTH, DAY_OF_WEEK ...
  ```
- Type parameters: single letter upper case
  ```java
  E (element) T (type) V (value type)
  ```

Initialization Block

- Statement block outside methods/constructors
- Executed before the body of any constructor

```java
without initialization block:
private long idNum;
private String name;
public Body() { }
public Body(String name, Star orbits) {
  this.name = name;
  this.orbits = orbits;
}

with initialization block:
private long idNum;
private String name;
private Star orbits;
private static long nextID = 0;
public Body() {
  idNum = nextID++;
}
public Body(String name, Star orbits) {
  this.name = name;
  this.orbits = orbits;
}
```

Static Initialization Block

- Similar to initialization block, but:
  - Can only reference static members
  - Executed only once, when class is first loaded

```java
public class Primes {
  private static int[] primes = new int[4];

  static {
    primes[0] = 2;
    for(int i = 1; i < primes.length; i++) {
      primes[i] = nextPrime(i);
    }
  }
}
```

// declaration of static nextPrime(int) ...

Summary

- Static members (ie class members)
  - Instance member belongs to one objects
  - Static member is shared amongst instances
- Enumerated types
- Packages (ie component catalogs)
  - Declaration
  - Another level of visibility
  - Import statements
  - Syntactic shorthand for frequent use
- Static imports
- Initialization blocks, including static initialization