Outline

- Static import
- Classpath
- Enumerations

`static` Import

- Normal import declarations import classes from packages, allowing them to be used without package qualification
- A `static` import declaration enables you to import the `static` members of a class (or interface) so you can access them via their unqualified names, i.e., without including class name and a dot (`.`)
  - `Math.sqrt(4.0)` → `sqrt(4.0)`

`static` Import

- Import a particular `static` member (single `static` import)
  - `import static packageName.ClassName.staticMemberName;`
  - The member could be a field or a method
- Import all `static` members of a class (`static` import on demand)
  - `import static packageName.ClassName.*;`
  - * is a wildcard, meaning "matching all"

Example

```
1  // Fig. 8.14: StaticImportTest.java
2  // Static import of Math class methods.
3  import static java.lang.Math.*;
4
5  public class StaticImportTest
6  {
7     public static void main( String[] args )
8     {
9        System.out.printf( "sqrt( 900.0 ) = %.1f\n", sqrt( 900.0 ) );
10       System.out.printf( "ceil( -9.8 ) = %.1f\n", ceil( -9.8 ) );
11       System.out.printf( "log( E ) = %.1f\n", log( E ) );
12       System.out.printf( "cos( 0.0 ) = %.1f\n", cos( 0.0 ) );
13    } // end main
14 } // end class StaticImportTest
```

Enables `Math` methods to be used by their simple names in this file

```
sqrt( 900.0 ) = 30.0
ceil( -9.8 ) = -9.0
log( E ) = 1.0
cos( 0.0 ) = 1.0
```

Importing a class

- A single-type-import declaration specifies one class to import
  - `import java.util.Scanner;`
- When your program uses multiple classes from the same package, you can import them with a type-import-on-demand declaration.
  - `import java.util.*; // import java.util classes`
- The wild card * informs the compiler that all `public` classes from the `java.util` package are available for use in the program.

Specifying Classpath (Compilation)

- When compiling a class that uses classes from other packages, `javac` must locate the `.class` files for all these classes.
- The compiler uses a special object called a class loader to locate the classes it needs.
  - The class loader begins by searching the standard Java classes that are bundled with the JDK.
  - Then it searches for optional packages.
  - If the class is not found in the standard Java classes or in the extension classes, the class loader searches the classpath, which contains a list of locations in which classes are stored

Specifying Classpath (Compilation)

- The classpath consists of a list of directories or archive files, each separated by a directory separator
  - Semicolon (`;`) on Windows, colon (`:`) on UNIX/Linux/Mac OS X
- Archive files are individual files that contain directories of other files, typically in a compressed format
  - Normally end with the `.jar` or `.zip` file-name extensions
- The directories and archive files specified in the classpath contain the classes you wish to make available to the compiler and the JVM

Specifying Classpath (Compilation)

- By default, the classpath consists only of the current directory
- The classpath can be modified by
  - providing the `-classpath (-cp)` option to the `javac` compiler
  - setting the `CLASSPATH` environment variable (not recommended).

```
javac -classpath .:/home/avh/classes:/usr/local/java/classes Test.java
```

Specifying Classpath (Execution)

- When you execute an application, the JVM must be able to locate the `.class` files of the classes used in that application.
- Like the compiler, the `java` command uses a class loader that searches the standard classes and extension classes first, then searches the classpath (the current directory by default).
- The classpath can be specified explicitly by using either of the techniques discussed for the compiler.

```
java -classpath .:/home/avh/classes:/usr/local/java/classes Test
```

Enumerations

- There are cases when a variable can only take one of a small set of predefined constant values, e.g., compass direction (N, S, E, W) and the days of a week (MON, TUE, etc.)
- In such cases, you should use an `enum` type to define a set of constants represented as unique identifiers

```
public enum Direction {
    NORTH, SOUTH, EAST, WEST
}
```

Enumerations

- `Direction` is a type called an **enumeration**, which is a special kind of class introduced by the keyword `enum` and a type name
- Inside the braces {} is a comma-separated list of enumeration constants, each representing a unique value (you don't need to care about the underlying implementation or the exact values)
- The identifiers in an `enum` must be unique

```
public enum Direction {
    NORTH, SOUTH, EAST, WEST
}
```

Enumerations

- Variables of the type `Direction` can be assigned only the four constants declared in the enumeration (other values are illegal, won't compile)
  - `Direction d = Direction.NORTH;`
- Like classes, all `enum` types are reference types

```
public enum Direction {
    NORTH, SOUTH, EAST, WEST
}
```

Enumerations

- Each `enum` declaration declares an `enum` class with the following restrictions:
  - `enum` constants are implicitly `final` (constants that shouldn't be modified)
  - `enum` constants are implicitly `static` (no objects need to access them)
  - Any attempt to create an object of an `enum` type with operator `new` results in a compilation error (constructor of an `enum` type can only be private or package-private, meaning without any access level modifier)
  - `enum` declarations contain two parts: (1) the `enum` constants, (2) the other members such as constructor, fields and methods (optional)
  - An `enum` constructor can specify any number of parameters and can be overloaded

Enumerations

- For every `enum`, the compiler generates the `static` method `values` that returns an array of the `enum`'s constants.
- When an `enum` constant is converted to a `String`, the constant's identifier is used as the `String` representation.

```
Direction d = Direction.NORTH;
System.out.println(d.toString()); // prints "NORTH"
```

Example

enum constants (objects in this example) initialized with constructor calls

```
public enum Book {
    JHTP("Java How to Program", "2012"),
    CHTP("C How to Program", "2007"),
    IW3HTP("Internet & World Wide Web How to Program", "2008"),
    CPPHTP("C++ How to Program", "2012"),
    VBHTP("Visual Basic 2010 How to Program", "2011"),
    CSHARPHTP("Visual C# 2010 How to Program", "2011");

    private final String title;
    private final String copyrightYear;
    private Book(String bookTitle, String year) {
        title = bookTitle;
        copyrightYear = year;
    }
    public String getTitle() {  return title;  }
    public String getCopyrightYear() {  return copyrightYear;  }
}
```

Just like normal classes, defining public methods for clients to use the enum type

Only six Book objects will be created, constants such as Book.JHTP store the references.

Example

```
import java.util.EnumSet;
    public class EnumTest {
    public static void main(String[] args) {
        System.out.println("All books:\n");

        for(Book book : Book.values())
            System.out.printf("%-10s%-45s%s\n", book,
                    book.getTitle(), book.getCopyrightYear());

        System.out.println("\nDisplay a range of enum constants:\n");

        for(Book book : EnumSet.range(Book.JHTP, Book.CPPHTP))
            System.out.printf("%-10s%-45s%s\n", book,
                    book.getTitle(), book.getCopyrightYear());
    }
}
```

`Values()` returns an array of the `enum`'s constants

`EnumSet`'s method `range()` returns a collection of the `enum` constants in the specified range of constants

Example

```
All books:

JHTP      Java How to Program                           2012
CHTP      C How to Program                              2007
IW3HTP    Internet & World Wide Web How to Program      2008
CPPHTP    C++ How to Program                            2012
VBHTP     Visual Basic 2010 How to Program              2011
CSHARPHTP Visual C# 2010 How to Program                 2011

Display a range of enum constants:

JHTP      Java How to Program                           2012
CHTP      C How to Program                              2007
IW3HTP    Internet & World Wide Web How to Program      2008
CPPHTP    C++ How to Program                            2012
```