 Enums
Enumerated values

• Sometimes you want a variable that can take on only a certain listed (enumerated) set of values

• Examples:
  – dayOfWeek: SUNDAY, MONDAY, TUESDAY, ...
  – month: JAN, FEB, MAR, APR, ...
  – gender: MALE, FEMALE
  – title: MR, MRS, MS, DR
  – appletState: READY, RUNNING, BLOCKED, DEAD

• The values are written in all caps because they are constants

• What is the actual type of these constants?
Enumerations

• In the past, enumerations were usually represented as integer values:
  – public final int SPRING = 0;
  – public final int SUMMER = 1;
  – public final int FALL = 2;
  – public final int WINTER = 3;

• This is a nuisance, and is error prone as well
  – season = season + 1;
  – now = WINTER; ...; month = now;

• Here’s the new way of doing it:
  – enum Season { WINTER, SPRING, SUMMER, FALL }
enums are classes

• An **enum** is actually a new type of class
  – You can declare them as inner classes or outer classes
  – You can declare variables of an enum type and get type safety and compile time checking
  – Each declared value is an instance of the enum class
  – **enums** are implicitly **public, static, and final**
  – You can compare **enums** with either **equals** or **==**

• **enums** extend **java.lang.Enum** and implement **java.lang.Comparable**
  – Hence, **enums** can be sorted

• **enums** override **toString** and provide **valueOf**
  – Example:
    – `Season season = Season.WINTER;`
    – `System.out.println(season); // prints WINTER`
    – `season = Season.valueOf("SPRING"); // sets season to Season.SPRING`
Advantages of the new enum

• Enums provide compile-time type safety
  – int enums don't provide any type safety at all: `season = 43;`
• Enums provide a proper name space for the enumerated type
• Enums are robust
  – If you add, remove, or reorder constants, you must recompile, and then everything is OK again
• Enum printed values are informative
  – If you print an int enum you just see a number
• Because enums are objects, you can put them in collections
• Because enums are classes, you can add fields and methods
Enums have weird constructors

- Except for constructors, an Enum is an ordinary class
- Each name listed within an Enum is actually a call to a constructor
- Example:
  - `enum Season { WINTER, SPRING, SUMMER, FALL }` //no semicolon
  - This constructs the four named objects, using the default constructor
- Eclipse example: DateExample.java, Month.java
 Enums extend and inherit from Enum

- `String toString()` returns the name of this `enum` constant, as contained in the declaration
- `boolean equals(Object other)` returns `true` if the specified object is equal to this `enum` constant
- `int compareTo(E o)` compares this `enum` with the specified object for order; returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object
- `static enum-type valueOf(String s)` returns the enumerated object whose name is `s`
- `static enum-type[] values()` returns an array of the enumeration objects