Enum Classes

ACS-1904 LECTURE 4

Enums

- It’s sometimes useful to be able to restrict the range of values that a variable is allowed to hold
  - This is, of course, beyond the existing definition of a variables range, int for example.
- The roll of a three-sided die for example.
  - It would be nice to be able to restrict the possible values to ONE, TWO, and THREE

Enums

- It would be nice to be able to restrict the possible values to ONE, TWO, and THREE
- We could use *int*, *char, or String.*
  - These don’t really restrict the range of possible values and are error-prone.

Enums

- Enum to the rescue:

```
41
42      public enum Season{WINTER, SPRING, SUMMER, FALL};
43
```

- A simple, maybe the simplest, example of an enumerated type

Enums

- Enum to the rescue:

```
48
STOP
50
51
Season timeOfYear = null;
timeOfYear = Season.FALL;
timeOfYear = Season.AUTUMN;
```

- Once the enum is declared it can be used like this

Enums

- Enum to the rescue:

```
48
STOP
50
51
Season timeOfYear = null;
timeOfYear = Season.FALL;
timeOfYear = Season.AUTUMN;
```

- A variable of type Season can only contain one of the four defined values.
- Does this sound like a '*data type*'?

Enums

- Enum to the rescue:

```
48  Season timeOfYear = null;
50  timeOfYear = Season.FALL;
51  timeOfYear = Season.AUTUMN;
```

- Line 50 works, FALL is one of the defined values for a variable of type Season
- Line 51 causes a syntax error. AUTMUN is not in the list.

Enums

To *enumerate* is:

- to name things one after another in a list
- This is clearly what is happing in the example on the preceding slides.
- We enumerated a list of seasons
  - Can you think of a couple of thousand more examples?

Enums

Enums are actually classes, but unlike the classes we are familiar with

- an enum comprises a finite set of instances
  - In our Seasons example, the Enum class had four instances, one for each of the constants SPRING, SUMMER, WINTER, and FALL
  - It is not possible to construct new instances of the class. i.e. no public constructor

Enums

- Enums were introduced in Java 5
- Instead of enums, legacy code will use `int`s that are defined as constants… but this approach is known to be error prone

Enumerated types

- An enumerated data type consists of a set of predefined values. You can use the data type to create variables that can hold only the values that belong to the enumerated data type.
- This gives us a measure of control over the data that our code will use to perform its computation
- Using Enum types enhances type safety by limiting the range of possible values that can be assigned to a given variable

Enumerated types

- An enumerated data type is more than just the range of values allowed
- You can also add:
  - Constructors:
    - are private it is a syntax error to declare an Enum constructor as public or protected
    - are only invoked when the enumerated constants are constructed.

Enumerated types

- An enumerated data type is more than just the range of values allowed
- You can also add:
  - Fields
    - Private, just like in the classes we are familiar with
    - Can provide additional information about the constant
    - For example, in our Seasons Enum, we could add a field description that could contain something like “Cold” for winter.

Enumerated types

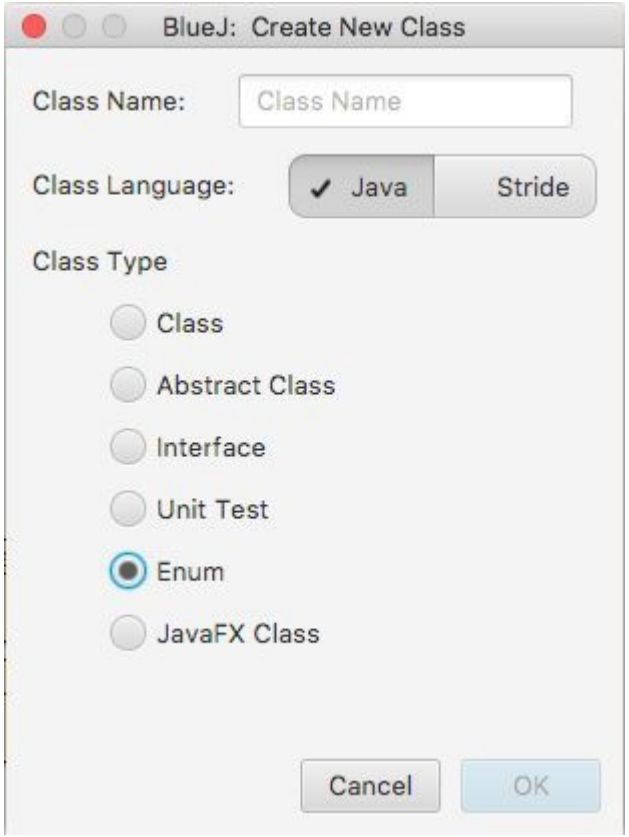
- An enumerated data type is more than just the range of values allowed
- You can also add:
  - Methods
    - Public so they can be accessed form outside the Enum itself i.e. in the driver code
    - In our example we could add a method to return the description

Examples

A few of the thousands of obvious examples:

- days of the week: Sunday, Monday,. . .Saturday
- months of the year: January, February, . . .December
- planets of the solar system: Mercury, Venus, . . .
- suits in a deck of cards: Spades, Hearts, Clubs, Diamonds
- card faces in a deck of cards: Ace, Deuce, Three, . . . King
- states of a door: open, closed
- four directions: north, south, east, west
- grades: A, B, C, D, F
- …

Creating an enum



Creating an enum - BlueJ

BlueJ generates a complete, but basic enum

- Modify this for your purpose

```
/**
 * Enumeration class Day - write a description …
 *
 * @author (your name here)
 * @version (version number or date here)
 */
public enum Day
{
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
}
```

Basic Enum

The enum Day results in 7 instances:

MONDAY:Day	FRIDAY:Day
TUESDAY:Day	SATURDAY:Day
WEDNESDAY:Day	SUNDAY:Day
THURSDAY:Day	

values()

The values() method returns the set of enum values as an array

- Given Day is an enum

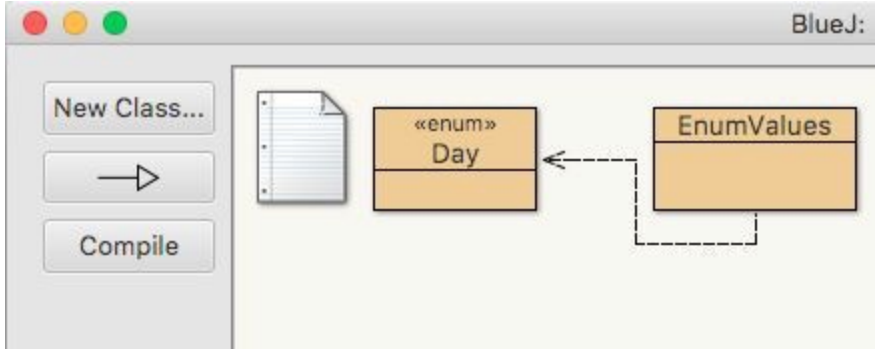
```
Day[]  mydays = Day.values();
```

- generates the array:

```
{Day.MONDAY, Day.TUESDAY, … Day.SUNDAY}
```

values() method

e.g. suppose EnumValues and Day are in same project:



values() method (EnumValues.java)

e.g. suppose EnumValues and Day are in same project:

```
public class EnumValues
{
    public static void main(String[] args){
        Day[] daysOfWeek = Day.values();
        System.out.println("Days of the week:");
        for (Day d: daysOfWeek)
            System.out.println(d);
    }
}
```

E.g. running EnumValues yields

```
Days of the week:
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY
SUNDAY
```

Java defines the `toString()` *method for enums to return the name of the enum*

Comparing enum values (CalculatePay.java)

As there is only one copy of any enum instance we can compare with ==

```
if (day == Day.SATURDAY || day == Day.SUNDAY )
    gross = 2* rate * hours ;
```

Gross pay doubled on weekends

valueOf() method

`valueOf(…)` takes one argument, a string value, and returns the enum instance of the same name.

`valueOf(…)` can be used to get the enum instance given its name equals the string argument

e.g.

```
Day.valueOf ("FRIDAY")
String today = kb.next();
Day.d = Day.valueOf (today.toUpperCase())
```

valueOf() method (CalculatePay.java)

Consider a method to calculate the gross pay as:

```
private static double calcPay(
        double rate, int hours, Day day){
            double gross;
            if (day==Day.SATURDAY || day==Day.SUNDAY)
                gross = 2*rate*hours;
            else
                gross = rate * hours;
            return gross;
        }
```

valueOf() method (CalculatePayForOneDay.java)

Here, we get the day from the user (a string), then get the enum, and then call calcPay

```
public class CalculatePayForOneDay
{
    public static void main(String[] args){
        Scanner kb = new Scanner(System.in);
        System.out.println("Enter rate hours day:");
        double rate = kb.nextDouble();
        int hours = kb.nextInt();
        String dayStr = kb.next().toUpperCase();
        Day day = Day.valueOf(dayStr);
        System.out.println("gross is $"
                    +calcPay(rate, hours, day));
    }
```

Get day from user

Get the enum

enum passed to calcPay

Enums are objects: data + methods

More advanced enums have data and methods.

- E.g.:
- Consider we need an enum for days of the week with a field to indicate if the day is a weekend or not.
  - Values for constructor are included in naming of instances.
  - Field(s) must be declared.
  - A constructor requires parameters of proper types.
  - Methods required for each field (typically a getter).

Example enum is called EnhancedDay

Enums are objects - give names and values

Data to be passed into an enum constructor is given in parentheses where the enums are named:

```
public enum EnhancedDay
{
   MONDAY(false), TUESDAY(false),
      WEDNESDAY(false), THURSDAY(false),
      FRIDAY(false), SATURDAY(true),
      SUNDAY(true);
```

values in ()

terminated with ;

Enum are objects: data fields

Field(s) declared :

```
public enum EnhancedDay
{
   MONDAY(false), TUESDAY(false),
      WEDNESDAY(false), THURSDAY(false),
      FRIDAY(false), SATURDAY(true),
      SUNDAY(true);

   private boolean weekend;
```

field

Enums are objects - constructor

Constructor is private

```
public enum EnhancedDay
{
    MONDAY(false), TUESDAY(false),
        WEDNESDAY(false), THURSDAY(false),
        FRIDAY(false), SATURDAY(true), SUNDAY(true);

    private boolean weekend;

    private EnhancedDay (boolean indicator){
        weekend = indicator;
    }
```

boolean values passed to constructor

constructor has parameter of proper type

Field assigned value

Enums are objects: constructor and objects

As the result of the constructor being called 7 times:

MONDAY:EnhancedDay
weekend=false

TUESDAY:EnhancedDay
weekend=false

WEDNESDAY:EnhancedDay
weekend=false

THURSDAY:EnhancedDay
weekend=false

FRIDAY:EnhancedDay
weekend=false

SATURDAY:EnhancedDay
weekend=true

SUNDAY:EnhancedDay
weekend=true

Enums are objects: methods(EnhancedDay.java)

Typically needs methods to get fields

```
public enum EnhancedDay
{
    MONDAY(false), TUESDAY(false),
        WEDNESDAY(false), THURSDAY(false),
        FRIDAY(false), SATURDAY(true), SUNDAY(true);

    private boolean weekend;

    private EnhancedDay (boolean indicator){
        weekend = indicator;
    }
    public boolean isWeekend(){
        return weekend;
    }
```

other objects can get field value

Using EnhancedDay(CalculatePayForEnhancedDay.java)

```
public class CalculatePayForEnhancedDay
{
    public static void main(String[] args){
        Scanner kb = new Scanner(System.in);
        System.out.println("Enter rate hours day:");
        double rate = kb.nextDouble();
        int hours = kb.nextInt();
        EnhancedDay day=
            EnhancedDay.valueOf(kb.next().toUpperCase());
        System.out.println("gross is $"
            +calcPay(rate, hours, day));
    }
```

Get enum

Get day from user

Get gross pay

Using EnhancedDay

```
private static double calcPay( double rate,
    int hours, EnhancedDay day) {
```

instance of EnhancedDay

```
    double gross;
```

isWeekend() is a method in EnhancedDay

```
    if (day.isWeekend()) //Double time on weekend
        gross = 2*rate*hours;
    else
        gross = rate * hours;
    return gross;
}
```

Enums are objects: summary

- Enums are never explicitly instantiated
  - cannot use new. (duh-doi, constructor is private)
- If there are fields, then values are included in parentheses when enum instances are named.
- Fields are declared in the enum.
- Fields are given values through the constructor (which is automatically called by the JVM).
- Methods can be included to get/set values included in an enum instance.