Enum

Yuliana Setiowati

Kebutuhan tipe baru

- sebelum J2SE 5.0, solusi untuk menangani masalah sekumpulan nilai konstanta, dicontohkan seperti di bawah ini: (jenis credit card yang bisa diterima oleh aplikasi)
  - public static final int VISA = 1;
  - public static final int MASTER_CARD = 2;
  - public static final int AMERICAN_EXPRESS = 3;

Kebutuhan tipe baru

- Permasalahan yang muncul :
  - tidak ada yang mengikat ketiga nilai menjadi semacam set dan kita bisa memberikan nilai yang salah pada variabel-variabel tersebut. Kondisi seperti ini disebut *not type safe* (tipe yang tidak aman).
- Kondisi ini dapat diperbaiki dengan membuat tipe yang relatif aman (tipe safe) dengan mendefinisikan suatu class, yaitu class AllowedCreditCard. Class tersebut mendefinisikan :
  - konstanta-konstanta di dalam kelas
  - variabel untuk menyatakan state object dari class tersebut.
  - Konstruktor private untuk mengeset state.

```
1. public class CreditCardTest {
2.  public static void main(String[] args){
3.   String creditCard = args[0].toUpperCase();
4.   if(creditCard.equals(AllowedCreditCard.VISA.getName())){
5.     System.out.println("Your credit card " + args[0] + " is accepted.");
6.   } else if (creditCard.equals(AllowedCreditCard.MASTER_CARD.getName())){
7.     System.out.println("Your credit card " + args[0] + " is accepted.");
8.    }else if
       (creditCard.equals(AllowedCreditCard.AMERICAN_EXPRESS.getName())){
9.       System.out.println("Your credit card " + args[0] + " is accepted.");
10.   } else {
11.         System.out.println("Sorry, we do not accept the credit card " +
            args[0] + " at this time.");
12.   }
13.  }
14. }

15. class AllowedCreditCard {
16.   protected final String card;
17.   public final static AllowedCreditCard VISA = new
          AllowedCreditCard("VISA");
18.   public final static AllowedCreditCard MASTER_CARD = new
         AllowedCreditCard("MASTER_CARD");
19.   public final static AllowedCreditCard AMERICAN_EXPRESS = new
         AllowedCreditCard("AMERICAN_EXPRESS");
20.   private AllowedCreditCard(String str){
21.      card=str;
22.   }
23.   public String getName(){
24.    return card;
25.   }
26. }
```

Output:

```
java CreditCardTest  visa
```

The output of this command will be

```
Your credit card visa is accepted.
```

Now assume that you issue the following command:

```
java CreditCardTest  discover
```

The output of this command will be

```
Sorry, we do not accept the credit card discover at this time.
```

Kebutuhan tipe baru

- Pada pendekatan ini, terdapat 3 state dari class AllowedCreditCard yang dinyatakan oleh tiga object yang dicreate dengan memberikan nilai yang berbeda pada variabel card.
- Karena konstruktor private, sehingga tidak bisa create object diluar class. Design seperti ini masih dianggap type safe.
- Tapi pada J2SE 5.0 terdapat solusi yang bagus dengan mengenalkan tipe baru yaitu enum.

Tipe Enum

- Tipe data enum dikenalkan di J2SE 5.0 berguna untuk variabel yang berisi sekumpulan nilai.
- Cara mendefinisikan variabel enum:
  - Mendefinisikan tipe enum dengan sekumpulan nilai.
  - Mendefinisikan variabel yang menyimpan satu dari nilai-nilai tersebut.
- Contoh:

```
enum AllowedCreditCard {VISA, MASTER_CARD,
    AMERICAN_EXPRESS};
AllowedCreditCard visa = AllowedCreditCard.VISA;
```

Contoh:

- Mendeklarasikan enum di luar class

```
enum CoffeeSize { BIG, HUGE, OVERWHELMING }  // this cannot be
                                             // private or protected

class Coffee {
   CoffeeSize size;
}

public class CoffeeTest1 {
   public static void main(String[] args) {
      Coffee drink = new Coffee();
      drink.size = CoffeeSize.BIG;         // enum outside class
   }
}
```

Contoh:

- Mendeklarasikan enum di dalam class

```
class Coffee2 {
  enum CoffeeSize  {BIG, HUGE, OVERWHELMING  }

  CoffeeSize size;
}

public class  CoffeeTest2 {
  public static void main (String[] args)  {
    Coffee2 drink = new Coffee2();
    drink.size = Coffee2.CoffeeSize.BIG;    // enclosing class
                                            // name required
  }
}
```

Contoh :

```
public class CoffeeTest1  {

  enum CoffeeSize { BIG, HUGE, OVERWHELMING }; // <--semicolon
                                               // is optional here
  public static void main(String[] args) {
    Coffee drink = new Coffee();
    drink.size = CoffeeSize.BIG;
  }
}
```

Tidak Legal

- Tidak bisa mendeklarasikan enum di dalam method

The following is NOT legal:

```
public class CoffeeTest1 {
  public static void main(String[] args) {
    enum CoffeeSize { BIG, HUGE, OVERWHELMING } // WRONG! Cannot
                                                // declare enums
                                                // in methods
    Coffee drink = new Coffee();
    drink.size = CoffeeSize.BIG;
  }
}
```

Enum

- Misalkan kita ingin menentukan nilai konstanta dari enum.
- Cara termudah dengan memberikan nilai enum (misal : BIG, HUGE, dan OVERWHELMING). Nilai enum sebagai object yang masing-masing mempunyai nilai instance variabel sendiri-sendiri.
- Nilai ini diberikan pada saat enum diinisialisasi, dengan memberikan nilai pada konstruktor enum.

```
enum CoffeeSize {

     BIG(8), HUGE(10), OVERWHELMING(16);
     // the arguments after the enum value are "passed"
     // as values to the constructor

     CoffeeSize(int ounces) {

       this.ounces = ounces; // assign the value to
                             // an instance variable
     }

    private int ounces;      // an instance variable each enum
                             // value has
    public int getOunces() {
      return ounces;
    }
}
```

```
class Coffee {
   CoffeeSize size;    // each instance of Coffee has-a
                       // CoffeeSize enum

   public static void main(String[] args) {
      Coffee drink1 = new Coffee();
      drink1.size = CoffeeSize.BIG;

      Coffee drink2 = new Coffee();
      drink2.size = CoffeeSize. OVERWHELMING;

      System.out.println(drink1.size.getOunces()); // prints 8
      System.out.println(drink2.size. getOunces()); // prints 16
   }
}
```

Point penting

- Konstruktor enum dijalankan secara otomatis. Contoh BIG(8) menjalankan konstruktor CoffeSize yang menerima parameter berupa int, dengan nilai 8.
- Konstruktor pada enum bisa lebih dari satu.

Contoh: Enum

```
enum ProgramFlags {
    showErrors(0x01),
    includeFileOutput(0x02),
    useAlternateProcessor(0x04);

    private int bit;

    ProgramFlags(int bitNumber)
    {
        bit = bitNumber;
    }

    public int getBitNumber()
    {
        return(bit);
    }

}
```

```
public class EnumBitmapExample {
    public static void main(String args[])
    {
        ProgramFlags flag = ProgramFlags.showErrors;

        System.out.println("Flag selected is: " +
                                flag.ordinal() +
                           " which is " +
                                flag.name());
    }
}
```

```
run:
Flag selected is: 0 which is showErrors
BUILD SUCCESSFUL (total time: 1 second)
```

Contoh :
Enum with switch statement

```
import java.util.*;

enum OperatingSystems {
    windows, unix, linux, macintosh
}

public class EnumExample1 {
    public static void main(String args[])
    {
        OperatingSystems os;

        os = OperatingSystems.windows;
        switch(os) {
            case windows:
                System.out.println("You chose Windows!");
                break;
            case unix:
                System.out.println("You chose Unix!");
                break;
            case linux:
                System.out.println("You chose Linux!");
                break;
            case macintosh:
                System.out.println("You chose Macintosh!");
                break;
            default:
                System.out.println("I don't know your OS.");
                break;
        }
    }
}
```

run:
You chose Windows!

Contoh:
Menggunakan Konstruktor Enum

```
enum Apple {
  A(10), B(9), C, D(15), E(8);

  private int price; // price of each apple

  // Constructor
  Apple(int p) { price = p; }

  // Overloaded constructor
  Apple() { price = -1; }

  int getPrice() { return price; }
}
```

Contoh :
Switch pada Enum dan Menambahkan Method di Enum

```
//Implementing Interfaces with Enums
enum GuitarFeatures implements Features {

  ROSEWOOD(0),          // back/sides
  MAHOGANY(0),          // back/sides
  ZIRICOTE(300),        // back/sides

  SPRUCE(0),            // top
  CEDAR(0),             // top

  AB_ROSETTE(75),       // abalone rosette
  AB_TOP_BORDER(400),   // abalone top border

  IL_DIAMONDS(150),     // diamond/square inlay
  IL_DOTS(0);           // dots inlays

  /** The upcharge for the feature */
  private float upcharge;

  GuitarFeatures(float upcharge) {
    this.upcharge = upcharge;
  }

  public float getUpcharge() {
    return upcharge;
  }
```

Contoh :
Switch pada Enum dan Menambahkan Method di Enum

```
  public String getDescription() {
    switch(this) {
      case ROSEWOOD:      return "Rosewood back and sides";
      case MAHOGANY:      return "Mahogany back and sides";
      case ZIRICOTE:      return "Ziricote back and sides";
      case SPRUCE:        return "Sitka Spruce top";
      case CEDAR:         return "Wester Red Cedar top";
      case AB_ROSETTE:    return "Abalone rosette";
      case AB_TOP_BORDER: return "Abalone top border";
      case IL_DIAMONDS:
        return "Diamonds and squares fretboard inlay";
      case IL_DOTS:
        return "Small dots fretboard inlay";
      default: return "Unknown feature";
    }
  }
}

interface Features {

  /** Get the upcharge for this feature */
  public float getUpcharge();

  /** Get the description for this feature */
  public String getDescription();

}
```

An enumeration of apple varieties.

```
public class EnumDemo {
  public static void main(String args[])
  {
    Apple ap;

    ap = Apple.C;

    // Output an enum value.
    System.out.println("Value of ap: " + ap);
    System.out.println();

    ap = Apple.B;

    // Compare two enum values.
    if(ap == Apple.B)
      System.out.println("ap conatins GoldenDel.\n");

    // Use an enum to control a switch statement.
    switch(ap) {
      case A:
        System.out.println("A is red.");
        break;
      case B:
        System.out.println("B is yellow.");
        break;
      case C:
        System.out.println("C is red.");
        break;
      case D:
        System.out.println("D is red.");
        break;
      case E:
        System.out.println("E is red.");
        break;
    }
  }
}
```

```
enum Apple {
   A,  B,  C,  D,  E
}
```

run:
Value of ap: C

ap conatins GoldenDel.

B is yellow.

Fungsi pada Enum

- public static Apple[] **values**()
  - Mengembalikan array yang berisi konstanta dari tipe enum, urutan sesuai pada saat pendeklarasian enum.
- public static Apple **valueOf**(String name)
  - Mengembalikan konstanta enum sesuai dengan inputan dari parameter String
- public final int **ordinal**()
  - Mengembalikan ordinal dari enum konstanta (dimulai dari 0)
- public final int **compareTo**(E o)
  - Membandingkan object enum dengan object enum lainnya berdasarkan urutan. Mengembalikan nilai negatif (object enum 1 < object enum 2), 0 (object enum 1 = object enum 2) dan positif (object enum 1 > object enum 2).

Contoh:
Use the built-in enumeration methods.

```
enum Apple {
  A, B, C, D, E
}

public class EnumDemo2 {
  public static void main(String args[])
  {
    Apple ap;

    System.out.println("Here are all Apple constants");

    // use values()
    Apple allapples[] = Apple.values();
    for(Apple a : allapples)
      System.out.println(a);

    System.out.println();

    // use valueOf()
    ap = Apple.valueOf("A");
    System.out.println("ap contains " + ap);

  }
}
```

run:
Here are all Apple constants
A
B
C
D
E

ap contains A

Use an enum constructor, instance variable, and method

```
enum Apple {
  A(10), B(9), C(12), D(15), E(8);
  private int price; // price of each apple
  // Constructor
  Apple(int p) {
    price = p;
  }
  int getPrice() {
    return price;
  }
}

public class EnumDemo3 {
  public static void main(String args[]) {
    Apple ap = Apple.A;

    // Display price of Winsap.
    System.out.println(ap.getPrice());

    // Display all apples and prices.
    System.out.println("All apple prices:");
    for (Apple a : Apple.values())
      System.out.println(a + " costs " + a.getPrice() + " cents.");
  }
}
```

```
run:
10
All apple prices:
A costs 10 cents.
B costs 9 cents.
C costs 12 cents.
D costs 15 cents.
E costs 8 cents.
```

Demonstrate ordinal(), compareTo(), and equals()

```
enum Apple {
  Jonathan, GoldenDel, RedDel, Winsap, Cortland
}

public class EnumDemo4 {
  public static void main(String args[])
  {
    Apple ap, ap2, ap3;

    // Obtain all ordinal values using ordinal().
    System.out.println("Here are all apple constants" +
                       " and their ordinal values: ");
    for(Apple a : Apple.values())
      System.out.println(a + " " + a.ordinal());

    ap =  Apple.RedDel;
    ap2 = Apple.GoldenDel;
    ap3 = Apple.RedDel;

    System.out.println();
```

Demonstrate ordinal(), compareTo(), and equals()

```
  // Demonstrate compareTo() and equals()
  if(ap.compareTo(ap2) < 0)
    System.out.println(ap + " comes before " + ap2);

  if(ap.compareTo(ap2) > 0)
    System.out.println(ap2 + " comes before " + ap);

  if(ap.compareTo(ap3) == 0)
    System.out.println(ap + " equals " + ap3);

  System.out.println();

  if(ap.equals(ap2))
    System.out.println("Error!");

  if(ap.equals(ap3))
    System.out.println(ap + " equals " + ap3);

  if(ap == ap3)
    System.out.println(ap + " == " + ap3);

  }
}
```

run:
Here are all apple constants and their ordinal values:
Jonathan 0
GoldenDel 1
RedDel 2
Winsap 3
Cortland 4

GoldenDel comes before RedDel
RedDel equals RedDel

RedDel equals RedDel
RedDel == RedDel

Creating an Enum

```
enum Grade { A, B, C, D, F, INCOMPLETE };

class Student {

  private String firstName;
  private String lastName;
  private Grade grade;

  public Student(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getFirstName() {
    return firstName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }

  public String getLastName() {
    return lastName;
  }

  public String getFullName() {
    return new StringBuffer(firstName)
            .append(" ")
            .append(lastName)
            .toString();
  }

  public void assignGrade(Grade grade) {
    this.grade = grade;
  }

  public Grade getGrade() {
    return grade;
  }
}
```

Creating an Enum

```
public class GradeTester {

  private Student student1, student2, student3;

  public GradeTester() {
    student1 = new Student("Brett", "McLaughlin");
    student2 = new Student("Ben", "Rochester");
    student3 = new Student("Dennis", "Erwin");
  }

  public void testGradeAssignment(PrintStream out) throws IOException {
    student1.assignGrade(Grade.B);
    student2.assignGrade(Grade.INCOMPLETE);
    student3.assignGrade(Grade.A);
  }

  public void listGradeValues(PrintStream out) throws IOException {
    Grade[] gradeValues = Grade.values();

    // for loop
    for (int i=0; i<gradeValues.length; i++) {
      out.println("Allowed value: '" + gradeValues[i] + "'");
    }

    // for/in loop
    for (Grade g : gradeValues ) {
      out.println("Allowed value: '" + g + "'");
    }
  }
```

```
public void testSwitchStatement(PrintStream out) throws IOException {
  StringBuffer outputText = new StringBuffer(student1.getFullName());

  switch (student1.getGrade()) {
    case A:
      outputText.append(" excelled with a grade of A");
      break;
    case B: // fall through to C
    case C:
      outputText.append(" passed with a grade of ")
                .append(student1.getGrade().toString());
      break;
    case D: // fall through to F
    case F:
      outputText.append(" failed with a grade of ")
                .append(student1.getGrade().toString());
      break;
    case INCOMPLETE:
      outputText.append(" did not complete the class.");
      break;
    default:
      outputText.append(" has a grade of ")
                .append(student1.getGrade().toString());
      break;
  }

  out.println(outputText.toString());
}
```

```
  public static void main(String[] args) {
    try {
      GradeTester tester = new GradeTester();

      tester.testGradeAssignment(System.out);
      tester.listGradeValues(System.out);
      tester.testSwitchStatement(System.out);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Output

- run:
- Allowed value: 'A'
- Allowed value: 'B'
- Allowed value: 'C'
- Allowed value: 'D'
- Allowed value: 'F'
- Allowed value: 'INCOMPLETE'
- Allowed value: 'A'
- Allowed value: 'B'
- Allowed value: 'C'
- Allowed value: 'D'
- Allowed value: 'F'
- Allowed value: 'INCOMPLETE'
- Brett McLaughlin passed with a grade of B
- BUILD SUCCESSFUL (total time: 0 seconds)