



Enumerated Types

תוכנה 1 בשפת ג'אווה - סמסטר א' תשס"ז
אוניברסיטת תל אביב





Enumerated Types



טיפוסים שכל מופעיהם קבועים וידועים מראש שכיחים מאוד בעולם התוכנה:

```
package cards.domain;
```

```
public class PlayingCard {
```



```
    // pseudo enumerated type
```

```
    public static final int SUIT_SPADES = 0;
```

```
    public static final int SUIT_HEARTS = 1;
```

```
    public static final int SUIT_CLUBS = 2;
```

```
    public static final int SUIT_DIAMONDS = 3;
```



```
    private int suit;
```

```
    private int rank;
```

```
    public PlayingCard(int suit, int rank) {
```

```
        this.suit = suit;
```

```
        this.rank = rank;
```

```
    }
```





Enumerated Types



```
public String getSuitName() {  
    String name = "";  
    switch (suit) {  
        case SUIT_SPADES:  
            name = "Spades";  
            break;  
        case SUIT_HEARTS:  
            name = "Hearts";  
            break;  
        case SUIT_CLUBS:  
            name = "Clubs";  
            break;  
        case SUIT_DIAMONDS:  
            name = "Diamonds";  
            break;  
        default:  
            System.err.println("Invalid suit.");  
    }  
    return name;  
}
```





Enumerated Types



● ואולם מימוש טיפוסים אלו בצורה זו אינו בטוח ויש לו חסרונות

```
package cards.tests;
```

```
import cards.domain.PlayingCard;
```

```
public class TestPlayingCard {  
    public static void main(String[] args) {
```

```
        PlayingCard card1 =  
            new PlayingCard(PlayingCard.SUIT_SPADES, 2);
```

```
        System.out.println("card1 is the " + card1.getRank() + " of "  
                            + card1.getSuitName());
```

```
        // You can create a playing card with a bogus suit.
```

```
        PlayingCard card2 = new PlayingCard(47, 2);  
        System.out.println("card2 is the " + card2.getRank() + " of "  
                            + card2.getSuitName());
```

```
    }
```

```
}
```





Enumerated Types



למימוש טיפוסים מניה בצורה זו כמה חסרונות:

● אינו שומר על בטיחות טיפוסים (Not typesafe)



● אינו שומר על מרחב שמות

● מימוש שברירי

● ערך ההדפסה אינו משמעותי





New Enumerated Types



- החל ב Java 5.0 התווסף לשפה המבנה enum
- הפותר את בעיית בטיחות הטיפוסים



```
package cards.domain;
```

```
public enum Suit {
```

```
    SPADES,
```

```
    HEARTS,
```

```
    CLUBS,
```

```
    DIAMONDS
```

```
}
```





New Enumerated Types



```
package cards.domain;
```

```
public class PlayingCard2 {
```

```
    private Suit suit;
```

```
    private int rank;
```



```
    public PlayingCard2(Suit suit, int rank) {
```

```
        this.suit = suit;
```

```
        this.rank = rank;
```

```
    }
```



```
    public Suit getSuit() {
```

```
        return suit;
```

```
    }
```





New Enumerated Types



```
public String getSuitName() {  
    String name = "";  
    switch (suit) {  
        case SPADES:  
            name = "Spades";  
            break;  
        case HEARTS:  
            name = "Hearts";  
            break;  
        case CLUBS:  
            name = "Clubs";  
            break;  
        case DIAMONDS:  
            name = "Diamonds";  
            break;  
        default:  
            assert false : "ERROR: Unknown type!";  
    }  
    return name;  
}
```





New Enumerated Types



```
package cards.tests;
```

```
import cards.domain.PlayingCard2;  
import cards.domain.Suit;
```

```
public class TestPlayingCard2 {  
    public static void main(String[] args) {
```

```
        PlayingCard2 card1 = new PlayingCard2(Suit.SPADES, 2);  
        System.out.println("card1 is the " + card1.getRank() +  
                            " of " + card1.getSuitName());
```

```
        // PlayingCard2 card2 = new PlayingCard2(47, 2);  
        // This will not compile.
```

```
    }  
}
```

ב Java כל דבר הוא עצם - על כן, הרחיבו גם את הקונספט של enum להיות מעין מחלקה (עם שדות, מתודות, בנאים...)

המבנה החדש פותר את בעיית הבטיחות אך לא את שאר בעיות



טיפוס מנייה כמחלקה



```
package cards.domain;
```

```
public enum Suit {  
    SPADES("Spades"),  
    HEARTS("Hearts"),  
    CLUBS("Clubs"),  
    DIAMONDS("Diamonds");
```



```
    private final String name;
```

```
    private Suit(String name) {  
        this.name = name;  
    }
```

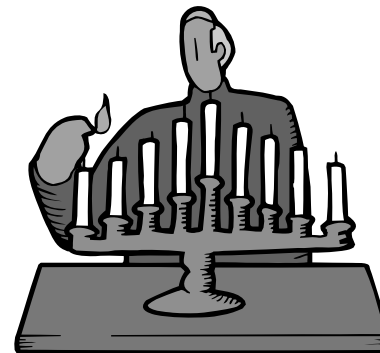


```
    public String getName() {  
        return name;  
    }
```

```
}
```



כעת אין צורך לשלוף את ייצוג
המחלקה כמחרוזת מבחוץ





שימוש בתכונות של טיפוס מניה



```
package cards.tests;
```

```
import cards.domain.PlayingCard2;  
import cards.domain.Suit;
```

```
public class TestPlayingCard3 {  
    public static void main(String[] args) {
```

```
        PlayingCard2 card1 = new PlayingCard2(Suit.SPADES, 2);  
        System.out.println("card1 is the " + card1.getRank() +  
                            " of " + card1.getSuit().getName());
```

```
        // NewPlayingCard2 card2 = new NewPlayingCard2(47, 2);  
        // This will not compile.
```

```
    }  
}
```





Polymorphic Behavior



```
public enum ArithmeticOperator {  
    // The enumerated values  
    ADD, SUBTRACT, MULTIPLY, DIVIDE;  
  
    // Value-specific behavior using a switch statement  
    public double compute(double x, double y) {  
        switch(this) {  
            case ADD:         return x + y;  
            case SUBTRACT:   return x - y;  
            case MULTIPLY:   return x * y;  
            case DIVIDE:     return x / y;  
            default:         throw new AssertionError(this);  
        }  
    }  
}
```





Polymorphic Behavior



```
public enum ArithmeticOperator {  
    ...
```



```
// Test case for using this enum
```

```
public static void main(String args[]) {  
    double x = Double.parseDouble(args[0]);  
    double y = Double.parseDouble(args[1]);  
    for(ArithmeticOperator op : ArithmeticOperator.values())  
        System.out.printf("%f %s %f = %f%n",  
                           x, op, y, op.compute(x,y));  
    }  
}
```





Real Polymorphism



```
public enum ArithmeticOperator2 {  
  
    ADD {  
        public double compute(double x, double y) {  
            return x + y;  
        }  
    },  
  
    SUBTRACT {  
        public double compute(double x, double y) {  
            return x - y;  
        }  
    },  
  
    MULTIPLY {  
        public double compute(double x, double y) {  
            return x * y;  
        }  
    },  
  
    DIVIDE {  
        public double compute(double x, double y) {  
            return x / y;  
        }  
    };  
  
    public abstract double compute(double x, double y);  
}
```





Bit Flags



● לעיתים לעצמים יש מס' מאפיינים/תכונות

● לגבי כל עצם יכולות להתקיים כל התכונות, חלקן

או אף אחת מהן



● למשל צורה גיאומטרית יכולה להיות

○קמורה, קעורה, מלאה, חלולה, ישרה, עגולה, צבעונית...

● איך הייתם מייצגים זאת?





Bit Flags



● דרך אחת – נשמור משתנה בוליאני לכל מאפיין

```
boolean isConvex;  
boolean isFull;  
...
```



● דרך שנייה – נשתמש ב-Bit Flags

```
int shapeAttributes;  
boolean isConvex() {  
    return shapeAttributes & 0x02;  
}
```





EnumSet



● בג'אוה 5 נוסף מימוש חדש ל-Set המבוסס על Enum



● כל הערכים בסט חייבים לבוא מ-Enum מוגדר כבר, או כזה המוגדר ביצירת הסט

● פנימית, הערכים מוחזקים כביטים, ז"א מאד יעילים





EnumSet



```
import java.util.*;
```

```
enum ShapeAttributes {
    CONVEX, FULL, STRAIGHT, COLORED
}
```



```
public class Testing {
    public static void main(String[] args) {
        Set<ShapeAttributes> s1 = EnumSet.of(ShapeAttributes.COLORED);
        if (s1.contains(ShapeAttributes.CONVEX))
            System.out.println("S1 is convex");
```



```
        Set<ShapeAttributes> s2 = EnumSet.of(ShapeAttributes.CONVEX,
        ShapeAttributes.FULL);
        if (s2.contains(ShapeAttributes.CONVEX))
            System.out.println("S2 is convex");
```

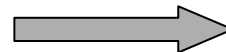


```
        Set<ShapeAttributes> s3 = EnumSet.allOf(ShapeAttributes.class);
        System.out.println(s3);
    }
```

● למשל לצורה שלנו...



S2 is convex



[CONVEX, FULL, STRAIGHT, COLORED]





EnumMap



אחיו החורג של EnumSet ●

```
enum Colors {  
    RED, GREEN, BLUE, YELLOW  
}
```



```
public class Testing {  
    public static void main(String[] args) {  
        Map<Colors,String> m = new EnumMap<Colors, String>(Colors.class);  
        m.put(Colors.RED, "Red");  
        m.put(Colors.BLUE, "Blue");  
        System.out.println(m);  
    }  
}
```





חג שמח!

