Enumerated Types

טיפוסי מנייה

שחר מעוז

תוכנה 1 בשפת ג'אוות

אוניברסיטת תל אביב

אורייבスタイル החל אביס
### Enumerated Types

Ciphers that are used in the code are known and unique:

```java
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;
    }
}
```
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

A disadvantage of using enumerated types is that you can easily forget to check that the
value you assign is valid.

```java
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

For enumerated types, there are several limitations:

1. Not typesafe
   - It does not maintain type safety.

2. No Name Space
   - It does not maintain name space.

3. Adding a new value to a complex type requires changes in multiple places.

   • Encapsulation

4. Has the type's structure
   - Has the structure of the type.

5. Newly added type loses its meaning.

6. Requires new symbols for new fields.

   •
New Enumerated Types

The `enum` type was introduced in Java 5.0 to provide a way of declaring sets of named values. This feature helps to resolve the issue of type safety.

```java
package cards.domain;

public enum Suit {
    SPADES,
    HEARTS,
    CLUBS,
    DIAMONDS
}
```
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;
    }
}
```java
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.PlayingCard;
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.
    }
}

-timeshade-

"NEW ENUMERATED TYPES"

package cards.tests;

import cards.domain.PlayingCard;
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.
    }
}
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 מבפנים
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.
    }
}

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);
        }
    }
}
public class SomeClient {
    ...

    // 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));
    }
}
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);
}
לעיתים לעצמים יש מפייین/תכונות
לגביל עצים יוכלות להתקיים כל התכונות, חלף או אף אחד

למשל צורה גיאומטרית יוכלת להיוות
• קמור, קעור, מלאה, חלול, ישרה, עדולה, צבעונית...

איך ניתן לייצג את这些东西? בקוד?

Bit Flags

לעייתם לעצמים יש מפייין/תכונות
לגביל עצים יוכלות להתקיים כל התכונות, חלף או אף אחד

למשל צורה גיאומטרית יוכלת להיוות
• קמור, קעור, מלאה, חלול, ישרה, עדולה, צבעונית...

איך ניתן לייצג את这些东西? בקוד?

Bit Flags

לעייתם לעצמים יש מפייין/תכונות
לגביל עצים יוכלות להתקיים כל התכונות, חלף או אף אחד

למשל צורה גיאומטרית יוכלת להיוות
• קמור, קעור, מלאה, חלול, ישרה, עדולה, צבעונית...

איך ניתן לייצג את这些东西? בקוד?
Bit Flags

Boolean isConvex;
boolean isFull;
...

int shapeAttributes;

public static final int fullMask = 0x01; // 000...0001
public static final int convexMask = 0x02; // 000...0010
public static final int straightMask = 0x04; // 000...0100

boolean isConvex() {
    return (shapeAttributes & convexMask) != 0;
}
EnumSet

- בג'ואה 5 נוסח Миיאוש חידש ל- Set המבוסס על numer.
- כל הערכים בסט יהיו שונים לסט- Enum מוגדר כבר, או כוה המוגדר ביצירת הסט
- פנימי, הערכים榭ודקים בביצים, ז"א מאדים צ'יימ

תוכה 1 בשפת ג'ואה
enum ShapeAttributes {

    FULL, CONVEX, 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 ליצור שולח...
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);
    }
}
סייגים

- טיפוסי מניה הן סוכרים תחביריים למחלים שאר של המופעים שלחן ידועים וනוצרן מראר

- בשונה משפות תכנות אחרות, טיפוס המניה הוא עצם לכל דבר, ובפרט יש לו שדות ושרותים

- בשונה משפות תכנות אחרות, טיפוס המניה הוא עצם לכל דבר, ובפרט יש לו שדות ושרותים

- מקרית פרגי של שמות טיפוסי מניה הוא עבורה ייצוג תכונות

- של טיפוס כלשהיה האשף אוסף התכונות ידוע מראות המחלקות של השימוש enumәפәתәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәәә עב