Introduction to Java (cs2514)

Lecture 10: The Joys of Enums

M. R. C. van Dongen

February 17, 2017

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Outline

- Many applications require groups of named constants.
- For example:
  - A suit of cards: `HEARTS`, `SPADES`, `CLUBS`, and `DIAMONDS`;
  - Predefined colours: `BLACK`, `WHITE`, `RED`, `BLUE`, ...;
  - And so on.
- In `Java` named constants are called `enum`s.
- They are the topic of this lecture.
  - We start with the `switch` statement.
    - This is a multi-way branching construct.
    - (Not really for `enum`s but needed for examples.)
  - We study a common, flawed pattern called *`int enums`*.
  - `Java` `enum`s overcome most of the flaws of `int` `enum`s.
  - `Java` `enum`s are just objects.
    - They may have state and common and specific behaviour.
- This lecture is partially based on [Bloch 2008, Item 30].
- Some of this lecture is based on the `Java` API documentation.

Multiway Branching

Java

```java
if (var == 0) {
    // First stuff
} else if (var == 1 || var == 3) {
    // Second stuff
} else if (var == 2 || var == 4) {
    // Third stuff
} ...
} else {
    // Final stuff
}
```

Java

```java
switch (var) {
case 0: // First stuff
case 1:
case 3: // Second stuff
case 2:
case 4: // Third stuff
...
default: // Final stuff
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

The `switch` Statement: Single Guards

Statements may end with `break`

Java

```
switch (⟨expr⟩) {
case ⟨constant #1⟩: ⟨statements #1⟩
case ⟨constant #2⟩: ⟨statements #2⟩
...
case ⟨constant #n⟩: ⟨statements #n⟩
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Multiple Guards

First Guards have Empty Statements

Java

```
switch (⟨expr⟩) {
case ⟨constant #1⟩:
case ⟨constant #2⟩:
...
case ⟨constant #m⟩: ⟨statements⟩
...
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

The switch Statement

Java

```
switch (⟨expr⟩) {
    case ⟨constant #1⟩: ⟨statements #1⟩
    case ⟨constant #2⟩: ⟨statements #2⟩
    ...
    case ⟨constant #n⟩: ⟨statements #n⟩
    default: ⟨default statements⟩
}
```

Example

Java

```
switch (character) {
    case 'A':
    case 'B':
    case 'C':
        System.out.println( "Range: A--C." );
        break;
    case 'e':
        System.out.println( "It's an 'e'" );
        break;
    default:
        System.out.println( "It's not in {A,B,C,e}" );
}
```


Outline

Multiway Branching

Int Enums

DIY

Enums to the Rescue

State and Behaviour

Specific Behaviour

Improvement

Strategy Enums

Use Attributes

The EnumSet Class

For Monday

Acknowledgements

Question Time

For Monday

Acknowledgements

References

About this Document

The `int`-`enum` Anti-Pattern

- An *enumerated type* represent a related set of constants.
  - The seasons of the year;
  - The suits in a deck of cards;
  - The graduation levles (pass, 2H2, 2H1, 1H);
  - ....
- Common, but flawed, implementation that uses `int` constants.

Don't Try This at Home

```
public static final int APPLE_FUJI     = 0;
public static final int APPLE_PIPPIN   = 1;

public static final int ORANGE_NAVEL   = 0;
public static final int ORANGE_TEMPLE  = 1;
public static final int ORANGE_BLOOD   = 2;
```

- This technique is called the *`int enum` pattern*.

Introduction to Java
M. R. C. van Dongen
Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

The `int-enum` Anti-Pattern

- An *enumerated type* represent a related set of constants.
  - The seasons of the year;
  - The suits in a deck of cards;
  - The graduation levles (pass, 2H2, 2H1, 1H);
  - ....
- Common, but flawed, implementation that uses `int` constants.

Don't Try This at Home

```
public static final int APPLE_FUJI     = 0;
public static final int APPLE_PIPPIN   = 1;

public static final int ORANGE_NAVEL   = 0;
public static final int ORANGE_TEMPLE  = 1;
public static final int ORANGE_BLOOD   = 2;
```

- This technique is called the *`int enum` pattern*.
- Never, ever, ever, use it.

Introduction to Java

M. R. C. van Dongen

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Int Enums are Flawed

Type safety: Int enums don't provide type safety.

Don't Try This at Home

```
if (APPLE_FUJI == ORANGE_BLOOD) {  }
int apple = ORANGE_BLOOD;
```

Maintainability: Programs with int enums are brittle.
- Int enums are compile-time constants.
- They are compiled into clients that use them.
- Client will break if enum constant changes.

Ease of use: Int enums are difficult to use.
- It is difficult to translate them to `Strings`.
- No reliable iteration over all allowed values.

Namespace: Int enum types have no private name space.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Int Enums are Flawed

Comparing Apples and Oranges??

Type safety: Int enums don't provide type safety.

Don't Try This at Home

```
if (APPLE_FUJI == ORANGE_BLOOD) { /* ?? */ }
int apple = ORANGE_BLOOD;
```

Maintainability: Programs with int enums are brittle.

- Int enums are compile-time constants.
- They are compiled into clients that use them.
- Client will break if enum constant changes.

Ease of use: Int enums are difficult to use.

- It is difficult to translate them to `Strings`.
- No reliable iteration over all allowed values.

Namespace: Int enum types have no private name space.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Int Enums are Flawed

Value Out of Range!

Type safety: Int enums don't provide type safety.

Don't Try This at Home

```
if (APPLE_FUJI == ORANGE_BLOOD) {  }
int apple = ORANGE_BLOOD;         // ??
```

Maintainability: Programs with int enums are brittle.

- Int enums are compile-time constants.
- They are compiled into clients that use them.
- Client will break if enum constant changes.

Ease of use: Int enums are difficult to use.

- It is difficult to translate them to `String`s.
- No reliable iteration over all allowed values.

Namespace: Int enum types have no private name space.

Int Enums are Flawed

Type safety: Int enums don't provide type safety.

Don't Try This at Home

```
if (APPLE_FUJI == ORANGE_BLOOD) {   }
int apple = ORANGE_BLOOD;
```

Maintainability: Programs with int enums are brittle.

- Int enums are compile-time constants.
- They are compiled into clients that use them.
- Client will break if enum constant changes.

Ease of use: Int enums are difficult to use.

- It is difficult to translate them to `String`s.
- No reliable iteration over all allowed values.

Namespace: Int enum types have no private name space.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Int Enums are Flawed

Type safety: Int enums don't provide type safety.

Don't Try This at Home

```
if (APPLE_FUJI == ORANGE_BLOOD) {  }
int apple = ORANGE_BLOOD;
```

Maintainability: Programs with int enums are brittle.

- Int enums are compile-time constants.
- They are compiled into clients that use them.
- Client will break if enum constant changes.

Ease of use: Int enums are difficult to use.

- It is difficult to translate them to `String`s.
- No reliable iteration over all allowed values.

Namespace: Int enum types have no private name space.

Introduction to Java

M. R. C. van Dongen

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Int Enums are Flawed

Type safety: Int enums don’t provide type safety.

Don’t Try This at Home

```
if (APPLE_FUJI == ORANGE_BLOOD) {  }
int apple = ORANGE_BLOOD;
```

Maintainability: Programs with int enums are brittle.

- Int enums are compile-time constants.
- They are compiled into clients that use them.
- Client will break if enum constant changes.

Ease of use: Int enums are difficult to use.

- It is difficult to translate them to `String`s.
- No reliable iteration over all allowed values.

Namespace: Int enum types have no private name space.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Implementing It Yourself

Java

```
public abstract class Beef {
    public static final Beef SHANK = new Beef( ) {
        @Override public double price( ) { return 1.0; }
    };
    public static final Beef SIRLOIN = new Beef( ) {
        @Override public double price( ) { return 2.0; }
    };
    public abstract double price( );

    private Beef( ) { }

    public static void main( String[] args ) {
        final Beef shank = Beef.SHANK;
        final Beef sirloin = Beef.SIRLOIN;
        ...
    }
}
```

Implementing It Yourself

Java

```
public abstract class Beef {
    public static final Beef SHANK = new Beef( ) {
        @Override public double price( ) { return 1.0; }
    };
    public static final Beef SIRLOIN = new Beef( ) {
        @Override public double price( ) { return 2.0; }
    };
    public abstract double price( );

    private Beef( ) { }

    public static void main( String[] args ) {
        final Beef shank = Beef.SHANK;
        final Beef sirloin = Beef.SIRLOIN;
        ...
    }
}
```

Implementing It Yourself

Java

```
public abstract class Beef {
    public static final Beef SHANK = new Beef( ) {
        @Override public double price( ) { return 1.0; }
    };
    public static final Beef SIRLOIN = new Beef( ) {
        @Override public double price( ) { return 2.0; }
    };
    public abstract double price( );

    private Beef( ) { }

    public static void main( String[] args ) {
        final Beef shank = Beef.SHANK;
        final Beef sirloin = Beef.SIRLOIN;
        ...
    }
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Implementing It Yourself

Java

```
public abstract class Beef {
    public static final Beef SHANK = new Beef( ) {
        @Override public double price( ) { return 1.0; }
    };
    public static final Beef SIRLOIN = new Beef( ) {
        @Override public double price( ) { return 2.0; }
    };
    public abstract double price( );

    private Beef( ) { }

    public static void main( String[] args ) {
        final Beef shank = Beef.SHANK;
        final Beef sirloin = Beef.SIRLOIN;
        ...
    }
}
```

A Serious Problem

Java

```java
public class MrEd extends Beef, implements Horse {
    @Override public double price( ) { return 0.2; }

    @Override public void talk( ) { ... }
}
```


Introduction to Java

M. R. C. van Dongen

Outline

Multiway Branching

Int Enums

DIY

Enums to the Rescue

State and Behaviour

Specific Behaviour

Improvement

Strategy Enums

Use Attributes

The EnumSet Class

For Monday

Acknowledgements

Question Time

For Monday

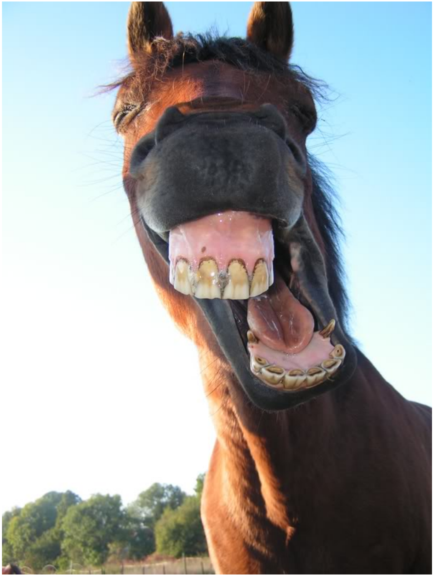
Acknowledgements

References

About this Document

A Serious Problem

Of Course



Outline

Multiway Branching

Int Enums

DIY

Enums to the Rescue

State and Behaviour

Specific Behaviour

Improvement

Strategy Enums

Use Attributes

The EnumSet Class

For Monday

Acknowledgements

Question Time

For Monday

Acknowledgements

References

About this Document

Java enums to the Rescue

- As of Release 1.5 Java provides the enum *type*.
- It overcomes most, if not all, shortcomings of int enums.

Java

```
public enum Apple  { FUJI, PIPPIN }
public enum Orange { NAVEL, TEMPLE, BLOOD }
```

- Each '`public enum` ⟨`class`⟩ { ⟨constants⟩ }' is a *class*.
- Each constant in ⟨constants⟩ is an instance of the class: an *object*.
- For each constant in any `enum` class, Java automatically defines one `public final` class attribute.
- Name of ⟨`constant`⟩ in ⟨`class`⟩ is ⟨`class`⟩.⟨`constant`⟩.
- All Java `enum` constructors are (implicitly) `private`.
- All instance methods are `final`, except for `toString( )`.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Why enums are Good

Type safety: Java enums are type safe.

Don't Try This at Home

```
if (Apple.FUJI == Orange.BLOOD) {  }
Apple apple = Orange.BLOOD;
```

Maintainability:
- enums aren't compiled as constants into clients.
- Rearranging values doesn't break clients.

Ease of use:
- Translating to Strings is easy: toString().
- Iterating over all enums is easy: values().

Namespace: Enum classes have a private name space.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Why enums are Good

Comparing Apples and Oranges??

Type safety: Java enums are type safe.

Don't Try This at Home

```
if (Apple.FUJI == Orange.BLOOD) { /* ?? */ }
Apple apple = Orange.BLOOD;
```

Maintainability:
- enums aren't compiled as constants into clients.
- Rearranging values doesn't break clients.

Ease of use:
- Translating to Strings is easy: toString().
- Iterating over all enums is easy: values().

Namespace: Enum classes have a private name space.

Why enums are Good

Value Out of Range!

Type safety: `Java` `enum`s are type safe.

Don't Try This at Home

```
if (Apple.FUJI == Orange.BLOOD) {  }
Apple apple = Orange.BLOOD;      // ??
```

Maintainability:
- `enum`s aren't compiled as constants into clients.
- Rearranging values doesn't break clients.

Ease of use:
- Translating to `String`s is easy: `toString()`.
- Iterating over all `enum`s is easy: `values()`.

Namespace: Enum classes have a private name space.

Why enums are Good

Type safety: Java enums are type safe.

Don't Try This at Home

```
if (Apple.FUJI == Orange.BLOOD) {  }
Apple apple = Orange.BLOOD;
```

Maintainability:
- enums aren't compiled as constants into clients.
- Rearranging values doesn't break clients.

Ease of use:
- Translating to Strings is easy: toString().
- Iterating over all enums is easy: values().

Namespace: Enum classes have a private name space.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Why enums are Good

Type safety: Java enums are type safe.

Don't Try This at Home

```
if (Apple.FUJI == Orange.BLOOD) {  }
Apple apple = Orange.BLOOD;
```

Maintainability:
- enums aren't compiled as constants into clients.
- Rearranging values doesn't break clients.

Ease of use:
- Translating to Strings is easy: toString().
- Iterating over all enums is easy: values().

Namespace: Enum classes have a private name space.

Outline

Multiway Branching

Int Enums

DIY

Enums to the Rescue

State and Behaviour

Specific Behaviour

Improvement

Strategy Enums

Use Attributes

The EnumSet Class

For Monday

Acknowledgements

Question Time

For Monday

Acknowledgements

References

About this Document

Why enums are Good

Type safety: Java enums are type safe.

Don't Try This at Home

```
if (Apple.FUJI == Orange.BLOOD) {  }
Apple apple = Orange.BLOOD;
```

Maintainability:
- enums aren't compiled as constants into clients.
- Rearranging values doesn't break clients.

Ease of use:
- Translating to Strings is easy: toString().
- Iterating over all enums is easy: values().

Namespace: Enum classes have a private name space.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Methods in `enum` Classes

`compareTo( that )`: Compares `this` `enum` with `that` for order.

`equals( that )`: Returns `true` if `this` `enum` equals `that`.

`hashCode( )`: Returns a hash code for `this` `enum`.

`toString( )`: Returns the name of this `enum` constant.

`name( )`: Returns the original name of this `enum`.

`ordinal( )`: Returns the *ordinal* of `this` `enum`.

Java Enums are Objects

- `int` enums only have a value.
- `Java` enums are objects.
  - They have state.
  - They have behaviour.
- Makes `Java` enums much more flexible.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

State and Behaviour

- Consider the eight planets of the solar system.
- Each planet has a mass and a radius.
- Using the mass and radius we compute the surface gravity.

Implementing the `Planet` Class

Java

```java
public enum Planet {
    MERCURY( 3.303e+23, 2.439e6 ),
    VENUS  ( 4.869e+24, 6.052e6 ),
    EARTH  ( 5.975e+24, 6.378e6 ),
    MARS   ( 6.419e+23, 3.393e6 ),
    JUPITER( 1.899e+27, 7.149e7 ),
    SATURN ( 5.685e+26, 6.027e7 ),
    URANUS ( 8.683e+25, 2.556e7 ),
    NEPTUNE( 1.024e+26, 2.477e7 );

    // Universal gravitational constant in m^3/kg s^2.
    private static final double G = 6.67300E-11;
    private final double mass;
    private final double radius;
    private final double gravity;

    Planet( double mass, double radius ) {
        this.mass = mass;
        this.radius = radius;
        gravity = G * mass / (radius * radius);
    }

    public double getMass( ) { return mass; }
    public double getRadius( ) { return radius; }
    public double getGravity( ) { return gravity; }
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Implementing the Planet Class

State

Java

```
public enum Planet {
    MERCURY( 3.303e+23, 2.439e6 ),
    VENUS  ( 4.869e+24, 6.052e6 ),
    EARTH  ( 5.975e+24, 6.378e6 ),
    MARS   ( 6.419e+23, 3.393e6 ),
    JUPITER( 1.899e+27, 7.149e7 ),
    SATURN ( 5.685e+26, 6.027e7 ),
    URANUS ( 8.683e+25, 2.556e7 ),
    NEPTUNE( 1.024e+26, 2.477e7 );

    // Universal gravitational constant in m^3/kg s^2.
    private static final double G = 6.67300E-11;
    private final double mass;
    private final double radius;
    private final double gravity;

    Planet( double mass, double radius ) {
        this.mass = mass;
        this.radius = radius;
        gravity = G * mass / (radius * radius);
    }

    public double getMass( ) { return mass; }
    public double getRadius( ) { return radius; }
    public double getGravity( ) { return gravity; }
}
```

Implementing the Planet Class

Behaviour

Java

```
public enum Planet {
    MERCURY( 3.303e+23, 2.439e6 ),
    VENUS  ( 4.869e+24, 6.052e6 ),
    EARTH  ( 5.975e+24, 6.378e6 ),
    MARS   ( 6.419e+23, 3.393e6 ),
    JUPITER( 1.899e+27, 7.149e7 ),
    SATURN ( 5.685e+26, 6.027e7 ),
    URANUS ( 8.683e+25, 2.556e7 ),
    NEPTUNE( 1.024e+26, 2.477e7 );

    // Universal gravitational constant in m^3/kg s^2.
    private static final double G = 6.67300E-11;
    private final double mass;
    private final double radius;
    private final double gravity;

    Planet( double mass, double radius ) {
        this.mass = mass;
        this.radius = radius;
        gravity = G * mass / (radius * radius);
    }

    public double getMass( ) { return mass; }
    public double getRadius( ) { return radius; }
    public double getGravity( ) { return gravity; }
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Let's Rock

Java

```
public class WeightTable {
    public static void main( String[] args ) {
        for (Planet planet : Planet.values( )) {
            double weight = surfaceWeight( planet, 1.0 );
            System.out.println( "1kg on " + planet
                              + " has a surface weight of "
                              + weight + "." );
        }
    }

    private static double surfaceWeight( final Planet planet, final double mass ) {
        return mass * planet.getGravity( );
    }
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Running the Application

Unix Session

```
$
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Running the Application

Unix Session

```
$ java WeightTable
```

Running the Application

Unix Session

```
$ java WeightTable
1kg on MERCURY has a surface weight of 3.7051525865812165.
1kg on VENUS has a surface weight of 8.870805573987766.
1kg on EARTH has a surface weight of 9.80144268461249.
1kg on MARS has a surface weight of 3.720666819023476.
1kg on JUPITER has a surface weight of 24.794508028173404.
1kg on SATURN has a surface weight of 10.443575504720215.
1kg on URANUS has a surface weight of 8.868889152162147.
1kg on NEPTUNE has a surface weight of 11.137021762915634.
$
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Specific Behaviour

- Our `Planet` application is very well behaved.
- All method results depend on input and attributes *only*.
- This is not always the case.
- For example, consider a calculator application.
  - There are four operations `PLUS`, `MINUS`, `TIMES`, and `DIVIDE`.
  - We'd like to apply operations to `double`s and get the result:
    - `double apply( double first, double second ).`
  - `assertTrue( 1.00 == PLUS.apply( 0.0, 1.0 ) ) &&` `assertTrue( -1.00 == MINUS.apply( 0.0, 1.0 ) ),` ....

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Specific Behaviour

- Our `Planet` application is very well behaved.
- All method results depend on input and attributes *only*.
- This is not always the case.
- For example, consider a calculator application.
  - There are four operations `PLUS`, `MINUS`, `TIMES`, and `DIVIDE`.
  - We'd like to apply operations to `double`s and get the result:
    - `double apply( double first, double second )`.
  - `assertTrue( 1.00 == PLUS.apply( 0.0, 1.0 ) ) &&` `assertTrue( -1.00 == MINUS.apply( 0.0, 1.0 ) )`, ...
  - The result *also* depends on the `enum` constant.

Outline

Multiway Branching

Int Enums

DIY

Enums to the Rescue

State and Behaviour

Specific Behaviour

Improvement

Strategy Enums

Use Attributes

The EnumSet Class

For Monday

Acknowledgements

Question Time

For Monday

Acknowledgements

References

About this Document

How do we Implement This?

Don't Try This at Home

```
public enum Operation {
    PLUS, MINUS, TIMES, DIVIDE;

    public double apply( double first, double second ) {
        double result;
        switch(this) {
        case PLUS:   result = first + second; break;
        case MINUS:  result = first - second; break;
        case TIMES:  result = first * second; break;
        case DIVIDE: result = first / second; break;
        default: String error = "Unknown Operation: " + this;
                 throw new AssertionError( error );
        }
        return result;
    }
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

How do we Implement This?

No!

Don't Try This at Home

```
public enum Operation {
    PLUS, MINUS, TIMES, DIVIDE, RECIPROCAL;

    public double apply( double first, double second ) {
        double result;
        switch(this) {
        case PLUS:   result = first + second; break;
        case MINUS:  result = first - second; break;
        case TIMES:  result = first * second; break;
        case DIVIDE: result = first / second; break;
        default: String error = "Unknown Operation: " + this;
                 throw new AssertionError( error );
        }
        return result;
    }
}
```

Constant-Specific Methods

Java

```
public enum Operation {
    PLUS   { @Override
             public double apply( double x, double y ) { return x + y; } },
    MINUS  { @Override
             public double apply( double x, double y ) { return x - y; } },
    TIMES  { @Override
             public double apply( double x, double y ) { return x * y; } },
    DIVIDE { @Override
             public double apply( double x, double y ) { return x / y; } };

    public abstract double apply( double first, double second );
}
```


Outline

Multiway Branching

Int Enums

DIY

Enums to the Rescue

State and Behaviour

Specific Behaviour

Improvement

Strategy Enums

Use Attributes

The EnumSet Class

For Monday

Acknowledgements

Question Time

For Monday

Acknowledgements

References

About this Document

Adding More Intuitive Printing

Java

```
public enum Operation {
    PLUS   { @Override
             public String toString( ) { return "+"; }
             @Override
             public double apply( double x, double y ) { return x + y; }},
  ⟨rest of class omitted⟩
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Using the Operation Class

Java

```
public class Calculator {
    public static void main( String[] args ) {
        final double first  = 6;
        final double second = 2;
        for (Operation op : Operation.values( )) {
            double result = op.apply( first, second );
            System.out.println( first + " " + op + " " + second
                                + " = " + result );
        }
    }
}
```

Unix Session

```
$
```


Introduction to Java

M. R. C. van Dongen

Outline

Multiway Branching

Int Enums

DIY

Enums to the Rescue

State and Behaviour

Specific Behaviour

Improvement

Strategy Enums

Use Attributes

The EnumSet Class

For Monday

Acknowledgements

Question Time

For Monday

Acknowledgements

References

About this Document

Using the `Operation` Class

Java

```
public class Calculator {
    public static void main( String[] args ) {
        final double first  = 6;
        final double second = 2;
        for (Operation op : Operation.values( )) {
            double result = op.apply( first, second );
            System.out.println( first + " " + op + " " + second
                                + " = " + result );
        }
    }
}
```

Unix Session

```
$ java Calculator
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Using the Operation Class

Java

```
public class Calculator {
    public static void main( String[] args ) {
        final double first  = 6;
        final double second = 2;
        for (Operation op : Operation.values( )) {
            double result = op.apply( first, second );
            System.out.println( first + " " + op + " " + second
                                + " = " + result );
        }
    }
}
```

Unix Session

```
$ java Calculator
6.0 + 2.0 = 8.0
6.0 - 2.0 = 4.0
6.0 * 2.0 = 12.0
6.0 / 2.0 = 3.0
$
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Getting Really Fancy Now??

Java

```
public enum Operation {
    PLUS {
        @Override
        public String toString( ) { return "+"; }
        @Override
        public double apply( double x, double y ) { return x + y; }
    }, MINUS {
        @Override
        public String toString( ) { return "-"; }
        @Override
        public double apply( double x, double y ) { return x - y; }
    }, TIMES {
        @Override
        public String toString( ) { return "*"; }
        @Override
        public double apply( double x, double y ) { return x * y; }
    }, DIVIDE {
        @Override
        public String toString( ) { return "/"; }
        @Override
        public double apply( double x, double y ) { return x / y; }
    };

    public abstract double apply( double first, double second );
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Factoring out Identical Behaviour

Java

```java
public enum Operation {
    PLUS( "+" ) {
        @Override
        public double apply( double x, double y ) { return x + y; }
    }, MINUS( "-" ) {
        @Override
        public double apply( double x, double y ) { return x - y; }
    }, TIMES( "*" ) {
        @Override
        public double apply( double x, double y ) { return x * y; }
    }, DIVIDE( "/" ) {
        @Override
        public double apply( double x, double y ) { return x / y; }
    };
    public abstract double apply( double first, double second );
    private final String symbol;

    Operation( String symbol ) {
        this.symbol = symbol;
    }

    @Override public String toString( ) { return symbol; }
}
```

Payroll Application

- Employees have a pay rate that depends on their grade.
- Our application gets the pay rate as its input.
- An employee's pay for a given day of the week is given by

$$\text{pay} = \text{base pay} + \text{overtime pay for that day}\,.$$

- The base pay is given by pay rate $\times$ hours worked.
- The overtime pay is given by

$$\text{overtime pay} = \text{pay rate} \times \text{overtime hours}/2\,.$$

Weekdays: Hours worked in excess of hours per shift (8).
Weekend: Hours worked on that day.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
A First Stab
Strategy Enum
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

First Stab at Implementation

Don’t Try This at Home

```
public enum SimplePayrollDay {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;

    private static final int HOURS_PER_SHIFT = 8;

    public double pay( double hoursWorked, double payRate ) {
        double basePay = hoursWorked * payRate;
        double overtimePay = overtimePay( hoursWorked, payRate );

        return basePay + overtimePay;
    }

    public double overtimePay( double hoursWorked, double payRate ) {
        double overtime;

        switch (this) {
        case SATURDAY:
        case SUNDAY: // Weekend
            overtime = hoursWorked;
            break;
        default:     // Weekday
            double difference = hoursWorked - HOURS_PER_SHIFT;
            overtime  = (difference < 0 ? 0 : difference);
        }
        return overtime * payRate / 2;
    }
}
```

What's Wrong?

- What if we add an extra type of day?
- For example, a Bank Holiday (special kind of Monday).
- We'd have to modify `overtimePay( )`.
- The application will break if we forget to make the change.

Introduction to Java

M. R. C. van Dongen

Outline

Multiway Branching

Int Enums

DIY

Enums to the Rescue

State and Behaviour

Specific Behaviour

Improvement

Strategy Enums

A First Stab

Strategy Enum

Use Attributes

The EnumSet Class

For Monday

Acknowledgements

Question Time

For Monday

Acknowledgements

References

About this Document

How to Fix It?

- We need different *strategies* for paying overtime.
- Strategy for `toString( )` in our computation is 100% shared.
- With the payrole application *some* strategies are shared, not all.
- Currently we have two strategies.
  - Each is *determined* by the kind of day: week days/weekend days.
  - The kind of day is a *property* of the day.
  - A property can be implemented as an *attribute*.
  - The attribute now *determines* the kind of day:
    - We can *compute* the kind of day from the attribute.
    - The kind of day *determines* the strategy.
    - Therefore, the attribute *determines* the strategy.
- We could implement our attribute as a `boolean`: `isWeekday`.
  - This would work now, but the requirements may change:
    - Double overtime rate for Christmas days?
- Probably better to have a *strategy* `enum` type.
  - The new strategy determines overtime pay computation.
- (Of course we implement it as an inner (`enum`) class.)

A Better Implementation

Java

```
public enum PayrollDay {
    SUNDAY(    PayType.WEEKEND ),
    MONDAY(    PayType.WEEKDAY ),
    TUESDAY(   PayType.WEEKDAY ),
    WEDNESDAY( PayType.WEEKDAY ),
    THURSDAY(  PayType.WEEKDAY ),
    FRIDAY(    PayType.WEEKDAY ),
    SATURDAY(  PayType.WEEKEND );

    private static final int HOURS_PER_SHIFT = 8;
    private final PayType type;

    PayrollDay( PayType type ) { this.type = type; }

    public double pay( double hoursWorked, double payRate ) {
        double basePay = hoursWorked * payRate;
        double overtimePay = type.overtimePay( hoursWorked, payRate );

        return basePay + overtimePay;
    }

    private enum PayType {
        WEEKEND { /* omitted. */ }, WEEKDAY { /* omitted. */ };
        public abstract
        double overtimePay( double hoursWorked, double payRate );
    }
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
A First Stab
Strategy Enum
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

The Details

Java

```
private enum PayType {
    WEEKEND {
        @Override
        public double overtimePay( double hoursWorked, double payRate ) {
            return hoursWorked * payRate / 2;
        }
    }, WEEKDAY {
        @Override
        public double overtimePay( double hoursWorked, double payRate ) {
            double difference = hoursWorked - HOURS_PER_SHIFT;
            double overtime   = (difference < 0 ? 0 : difference);
            return overtime * payRate / 2;
        }
    };
    public abstract
    double overtimePay( double hoursWorked, double payRate );
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
A First Stab
Strategy Enum
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Why Strategy enums are Good for You

- The overtime pay computation is *what varies*.
- The strategy enum *isolates* what varies.
- *Localises* the code for overtime pay computation.
- *Global* change in rules translates to *local* change in code:
  - Easy to remove days and strategies.
  - Easy to change strategies.
  - Easy to add new days for existing strategies.
  - Easy to add new days and new strategies.

Java

```
public enum PayrollDay {
    ...
    BANK_HOLIDAY( PayType.BANK_HOLIDAY ),
    ...
    private enum PayType {
        ...
        BANK_HOLIDAY {
            @Override
            public double overtimePay( double hoursWorked, double payRate ) {
                return hoursWorked * payRate;
            }
        ...
    }
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
A First Stab
Strategy Enum
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

A Music Application

Don't Try This at Home

```
public enum Ensemble {
    SOLO,   DUET,   TRIO,  QUARTET, QUINTET,
    SEXTET, SEPTET, OCTET, NONET,   DECTET;

    public int size( ) { return 1 + ordinal( ); }
}
```

This class will break if:

- Constants are re-ordered.
- Constants are removed.
- Constants are added and there are "holes."
- Constants are added with the same size as existing ensembles.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

A Better Approach

Java

```
public enum Ensemble {
    SOLO( 1 ), DUET( 2 ), TRIO( 3 ), QUARTET( 4 ),
    QUINTET( 5 ), SEXTET( 6 ), SEPTET( 7 ), OCTET( 8 ),
    DOUBLE_QUARTET( 8 ), NONET( 9 ), DECTET( 10 );
    private final int size;

    private Ensemble( final int size ) {
        this.size = size;
    }

    public int size( ) {
        return size;
    }
}
```

- Order can be changed.
- Constants can be removed.
- Constants can be added.

A Better Approach

Java

```
public enum Ensemble {
    SOLO( 1 ), DUET( 2 ), TRIO( 3 ), QUARTET( 4 ),
    QUINTET( 5 ), SEXTET( 6 ), SEPTET( 7 ), OCTET( 8 ),
    DOUBLE_QUARTET( 8 ), NONET( 9 ), DECTET( 10 );
    private final int size;

    private Ensemble( final int size ) {
        this.size = size;
    }

    public int size( ) {
        return size;
    }
}
```

- Order can be changed.
- Constants can be removed.
- Constants can be added.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Some Bitwise Operators

`lhs << rhs` Shift the `int lhs` to the left by `rhs` bits:[1]

- `(1 << 1) == 2;`
- `(2 << 2) == 8;`
- `(3 << 32) == 3;`

`~operand` Complement of `operand`:

- `(~0) == -1;`
- `(~1) == -2;`
- `(~-1) == 0;`

`lhs & rhs` Bitwise and of `lhs` and `rhs`:

- `(7 & 3) == 3;`
- `(16 & 15) == 0;`
- `(32 & 31) == 0;`

`lhs | rhs` Bitwise or of `lhs` and `rhs`:

- `(7 | 3) == 7;`
- `(4 | 3) == 7;`
- `(32 | 31) == 63;`

[1] Only the last 5 bits of `rhs` are used for the shift operation.

The *Bitwise Field Enumeration* Anti-Pattern

Java

```
public class TextStyle {
    public static final int STYLE_BOLD      = 1 << 0;
    public static final int STYLE_ITALIC    = 1 << 1;
    public static final int STYLE_UNDERLINE = 1 << 2;
    ...
    private int style = 0;

    public void computeUnion( int otherStyle ) {
        style |= otherStyle;
    }

    public void computeDifference( int otherStyle ) {
        style &= ~otherStyle;
    }

    public boolean containsStyle( int otherStyle ) {
        return otherStyle == (style & otherStyle);
    }
}
```

Disadvantages

- All disadvantages of bit-enum anti-pattern.
- Doesn't work if set has more than 32 members.

EnumSet to the Rescue

Java

```
import java.util.*;

public class TextStyle {
    public enum Style { BOLD, ITALIC, UNDERLINE }
    private EnumSet<Style> style
        = EnumSet.copyOf( new HashSet<Style>( ) );

    ...
}
```


Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

EnumSet to the Rescue (Continued)

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Java

```
public void computeUnion( EnumSet<Style> otherStyle ) {
    // addAll inherited from Set
    style.addAll( otherStyle );
}

public void computeDifference( EnumSet<Style> otherStyle ) {
    // removeAll inherited from AbstractSet
    style.removeAll( otherStyle );
}

public boolean containsStyle( EnumSet<Style> otherStyle ) {
    // containsAll inherited from AbstractCollction.
    return style.containsAll( otherStyle );
}
```

For Monday

- Study the presentation, and
- [Bloch 2008, Item 30] if you have the book.
- Do not use enumerated types for the assignment 2.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Acknowledgements

- This lecture is partially based on [Bloch 2008, Item 30].
- This lecture is also based on the Java API documentation.

Question Time

Questions Anybody?

For Monday

- Study the presentation, and
- [Bloch 2008, Item 30] if you have the book.
- Do not use enumerated types for the assignment 2.

Acknowledgements

- This lecture is partially based on [Bloch 2008, Item 30].
- This lecture is also based on the Java API documentation.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The EnumSet Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document

Bibliography I

Bloch, Joshua [2008]. *Effective Java*. Addison–Wesley. ISBN: 978-0-321-35668-0.

About this Document

- This document was created with `pdflatex`.
- The LaTeX document class is `beamer`.

Outline
Multiway Branching
Int Enums
DIY
Enums to the Rescue
State and Behaviour
Specific Behaviour
Improvement
Strategy Enums
Use Attributes
The `EnumSet` Class
For Monday
Acknowledgements
Question Time
For Monday
Acknowledgements
References
About this Document