



# Allt om Enum

Rikard Thulin, Java evangelist & senior konsult på B3IT

Presentationen finns på min blogg: <http://thulin.pp.se/rikard>

# Enum för dummys

En enum typ är en typ vars fält (variabler) består av en mängd av konstanter

```
enum Simpsons { HOMER,  
                MARGE,  
                BART,  
                LISA,  
                MAGGIE };
```

# Int enum (anti) pattern

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

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

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

# Int enum (anti) pattern

- Brister

// En familj?

```
int i = (MARGE + HOMER) * BART;
```

- Svårtolkat när man debuggar

# Användningsfall

- Använd alltid enum när du har ett fixt mängd av konstanter
- Exempel
  - ▶ veckans dagar, kort i en kortlek
  - ▶ men även för mängder där du vid kompileringstillfället känner till alla värden
  - ▶ menyval, kommandorads alternativ, osv

# Java enum

- Enum är en typ i Java
- Enum är inte en sockrad int
- Enum + switch = sant
- `for (Simpsons s : Simpsons.values())`
- `EnumA == EnumB` funkar

# Stil guide

- Namnet på en enum typ följer samma regler som för klasser, dvs stor begynnelsebokstav
- Värdena i enum typen skrivs med **VERSALER** då de är konstanter

# Snabbis

- Om du inte använder Java 1.5 än  
(troligen någon slags bestraffning - gör bot!)
- Koda inte:  
Enumeration enum = vector.elements();

# Vad är en Java enum

- En enum är en speciell klass
- Ärver `java.lang.Enum` implicit
- Kan därför inte ärva något annan klass
- Konstruktörerna måste vara paket-privata eller privata
- Konstruktörerna kan inte anropas externt

# Vad är en Java enum

- Implementerar Comparable
- Implementerar Serializable
- Kan inte ha finalizer (kompileringsfel)
- Kan inte vara final (är det implicit)
- Kan deklareras i innerklasser, men är implicit static
- Kan ha metoder

# Initialisering

*The Java Language Specification:*

“It is a compile-time error to reference a static field of an enum type that is not a compile-time constant (§15.28) from constructors, instance initializer blocks, or instance variable initializer expressions of that type. It is a compile-time error for the constructors, instance initializer blocks, or instance variable initializer expressions of an enum constant *e* to refer to itself or to an enum constant of the same type that is declared to the right of *e*”

# Prestanda

- Ej märkbar effekt (förutom i brödrostar..?)
- Initiering av en enum tar dock tid och plats
- Annars ungefär som int konstanter

```
public enum Bok {  
    EFFECTIVE_JAVA(346, 0321356683),  
    GWT_IN_PRACTICE(450, 1933988290);
```

```
    private final int sidor;  
    private final long isbn;
```

```
    Bok(int sidor, long isbn) {  
        this.sidor = sidor;  
        this.isbn = isbn;  
    }
```

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

# Exempel

```
for (Bok bok : Bok.values()) {  
    System.err.printf("Antal sidor %s: %d%n", bok, bok.sidor());  
}
```

....

// Utskriften blir

Antal sidor EFFECTIVE\_JAVA: 346

Antal sidor GWT\_IN\_PRACTICE: 450

# Tips

- Gör en enum för att översätta text

```
public enum Text {  
    LOGIN("login"), LOGIN_ERROR("err");  
    private final String key;  
    ...  
    public String getString(Locale locale) {  
        return resource.getString(key, locale);  
    }  
}
```

# Tips - version 2

```
public enum Text {
    LOGIN("login"), LOGIN_ERROR("err");
    private String key;
    private List<WeakReference<ThreadLocal<Locale>>> =..

    public void setLocale(ThreadLocal<Locale> l) { .. }
    private Locale getLocale() { .. }
    public String toString() {
        return resource.getString(key, getLocale());
    }
}
```

# “abstrakt” enum

```
public void diagnos(Bil bil) {  
    switch (bil) {  
        case SAAB: doSaabDiagnos(); break;  
        case VOLVO: doVolvoDiagnos(); break;  
        default: throw new AjajajException();  
    }  
}
```

// Tänkt om någon lägger till BMW..

# “abstrakt” enum

```
public enum Bil {  
    SAAB { public void diagnos() { .. },  
    VOLVO { public void diagnos() { ... };  
  
    public abstract void diagnos();  
}
```

# Enum ordinals

- Metoden `ordinal()` returnerar positionen av en enum
- Om du lägger till ett värde till en enum så ändras ordinal
- Man kan inte använda samma värde flera gånger

```
public enum Obra {  
    ENKEL, TOR;  
    public int biljetter() { return ordinal()+ 1);  
}
```

# Enum ordinals

- Använd inte ordinal()!  
“Most programmers will have no use for this method. It is designed for use by general-purpose enumbased data structures such as EnumSet and EnumMap”

```
public enum Bra {  
    TIOKORT(10), ENKEL(1), TOR(2);  
    private final int biljetter;  
    Bra(int biljetter) { this.biljetter=biljetter; }  
    public int biljetter() { return biljetter; }  
}
```

# Serializable

- The process by which enum constants are serialized cannot be customized: any class-specific `writeObject`, `readObject`, `readObjectNoData`, `writeReplace`, and `readResolve` methods defined by enum types are ignored during serialization and deserialization. Similarly, any `serialPersistentFields` or `serialVersionUID` field declarations are also ignored--all enum types have a fixed `serialVersionUID` of 0L.

# Singleton

- “a single-element enum type is the best way to implement a singleton” - Joshua Bloch

```
public enum Logger {  
    INSTANCE;  
  
    public void info(String message) { ... }  
}
```

# Varför?

- Om en “vanlig” singleton (måste) implementerar Serializable så skapas en ny instans varje gång objektet deserializeras
- Eller sätt alla fält som “transient” och implementera metoden readResolve()
- En privilegerad klient kan skapa anropa en privat konstruktor mha reflection
- Enum singleton har inte dessa problem!

# Men...

- Man kan naturligtvis skapa flera instanser av en enum (läs alla klasser) genom att ladda dom i olika klasssladdare -- men det är en feature :-)

# EnumSet

- Äntligen slipper vi:

```
public class Text {  
    public static final int STIL_FET      = 1<<0;  
    public static final int STIL_KURSIV  = 1<<1;  
    public static final int STIL_USTRK   = 1<<2;  
  
    public void applicera(int stilar) { ... }  
}
```

```
// bitvis OR
```

```
test.applicera(STIL_FET | STIL_KURSIV);
```

# EnumSet

- Implementerar Set
- Representeras internt av en bit vektor
- Representeras av en enda long ( $\leq 64$  enums)
- removeAll / retainAll är implementerad med hjälp av binär aritmetik
  - finns inte immutable EnumSet (kanske i 1.7)
    - Collections.unmodifiableSet, dock sämre prestanda...

# EnumSet exempel

```
public class Text {  
    public enum Stil { FET,  
                      KURSIV,  
                      UNDERSTRYKNING }  
  
    public applicera(Set<Stil> stilar) { ... }  
}  
  
// Exempel  
text.applicera(EnumSet.of(Stil.FET, Stil.KURSIV));
```

# EnumSet Tips

```
enum Day { MONDAY, TUESDAY, WEDNESDAY,  
THURSDAY, FRIDAY, SATURDAY, SUNDAY };
```

```
for (Day d : EnumSet.range(Day.MONDAY, Day.FRIDAY))  
    System.out.print(d + " ");
```

```
// Utskrift
```

```
MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY
```

# EnumMap

- Alla nycklar måste komma från samma enum typ
- Nycklarna behåller naturliga ordning (ordningen som konstanterna är deklarerade)

# Sammanfattning

- Betydligt mer läsbara än int konstanter
- Typsäkra
- Funkar med switch
- För Singletons
- EnumSet
- Vissa egenskaper (metoder) bör används restriktivt - eller i alla fall genomtänkt

# Frågor & svar

Presentationen finns på min blogg: <http://thulin.pp.se/rikard>

# Referenser

- <http://thulin.pp.se/rikard>
- <http://blog.nominet.org.uk/tech/2008/04/01/java-enum-methods-serialisation-fun/>
- <http://www.ajaxonomy.com/2007/java/making-the-most-of-java-50-enum-tricks>
- <http://www.javaworld.com/javaworld/jvatips/jw-jvatip122.html>
- <http://java.sun.com/javase/6/docs/platform/serialization/spec/serial-arch.html#6469>
- [http://java.sun.com/docs/books/jls/third\\_edition/html/classes.html#8.9](http://java.sun.com/docs/books/jls/third_edition/html/classes.html#8.9)