DD2385

Programutvecklingsteknik

Några bilder till föreläsning 11

12/5 2014

Innehåll

- Enum m.m.
- OOA (ObjektOrienterad Analys)
- Utvecklingsmetodik
  - särskilt XP-liknande

Uppräkningstyper – enum

Definiera egen typ med *egna* värden:

```
enum Day {Monday, Tuesday, Wednesday,
          Thursday, Friday, Saturday,
          Sunday}
```

Day är en klass med konstanta värden.
Värdena refereras med

`Day.Wednesday` `Day.Saturday`

```
Day theDay = Day.Saturday;
Day[] specialdays =
{Day.Tuesday, Day.Thursday, Day.Sunday};
```

Lägg i filen **Day.java**

Uppräkningstyper – enum

values() alla värden inom typen:

```
for (Day d: Day.values())
    System.out.println(d);
```

ger utskrift

```
Monday
Tuesday
Wednesday
Thursday
...
```

Uppräkningstyper – enum

ordinal() ger ordningsnummer

```
for (Day d: Day.values())
    System.out.print(d.ordinal() + " ");
```

ger utskrift

```
0 1 2 3 4 5 6
```

Går att jämföra och sortera: implements Comparable

```
System.out.println
  (Day.Thursday.compareTo(Day.Saturday));
```

ger utskrift

```
-2
```

Spelkort med enum

```
class Spelkort {
   Farg farg;
   Valor valor;

   Spelkort(Farg f, Valor v){
      farg = f;
      valor = v;
   }

   public String toString () {
      return farg + "-" + valor;
   }
}
```

Spelkort med enum

enum för Farg

```
public enum Farg
  {HJARTER, KLOVER, RUTER, SPADER}
```

enum för Valor

```
public enum Valor
  {ESS, TVA, TRE, FYRA, FEM, SEX,
   SJU, ATTA, NIO, TIO, KNEKT,
   DAM, KUNG}
```

Kortlek

```
class Kortlek {
  Spelkort[] lek = new Spelkort[52];

  Kortlek() {
    int i=0;
    for (Farg farg: Farg.values())
      for (Valor valor:Valor.values())
        lek[i++] =
          new Spelkort(farg,valor);
  }
}
```

Metoder att lägga till i Kortlek : toString()

En lång String med 5 kort per rad

```
public String toString() {
    StringBuilder allt = new StringBuilder();
    int rad = 0;
    for (Spelkort s: lek){
        allt.append(s + " ");
        if (rad++ == 4){
            allt.append("\n");
            rad = 0;
         }
    }
    allt.append("\n");
    return allt.toString();
}
```

Blanda korten

Arrays.asList(kortlek)

gör lista intimt kopplad till en array (här kortlek).

Ändringar i listan görs även i arrayen.

```
Collections.shuffle(Arrays.asList(kortlek));
```

Listan blandas → Kortleken blandas

Metoder att lägga till i Kortlek : blanda()

```
void blanda() {
    Collections.shuffle(Arrays.asList(lek));
}
```

Testa kortleken: skapa, skriv ut, blanda, skriv ut

```
public static void main(String[] u) {
    Kortlek lek = new Kortlek();
    System.out.println("Kortleken fran borjan:");
    System.out.println(lek);
    lek.blanda();
    System.out.println("\nKortleken blandad:");
    System.out.println(lek);
}
```

Gamla Spelkort

```
class Spelkort {
    String farg;
    int valor;

    Spelkort(String f, int v){
        farg = f;
        valor = v;
    }

    public String toString () {
        return farg + "-" + valor;
    }
}
```

Varför är nya Spelkort bättre än gamla ?

- Med enum så är Farg och Valor väldefinierade
- Största och minsta värden är säkert tillgängliga
  values() ordinal()
- Med gamla Spelkort kan man skapa t.ex.
  new Spelkort("VIRUS",-1729)

Enum är mer än enstaka värden

- Enum är klass med konstruktor och metoder (om man vill)
- Enum - klasser är `final`, går ej att ärva från
- Enum med ett enda värde → `Singleton`

```
public enum Singleton {THEONLY}
```

Namnen väljs förstås som man vill

```
public enum Earth {ourPlanet}
```

```
public enum Koh_i_noor {Diamond}
```

Singleton med lite innehåll

```
public enum SnowWhite {
  THEONLY;
  int age;  boolean married = false;
  Object spouse;

  void incrAge () {age++;}

  void marry (Object newPartner) {
    married = true; spouse = newPartner;
  }

  public String toString() {
    String rstring = "SnowWhite " + age;
    if (married)
      rstring += ", married to " + spouse;
    return rstring;
  }}
```

Sju dvärgar

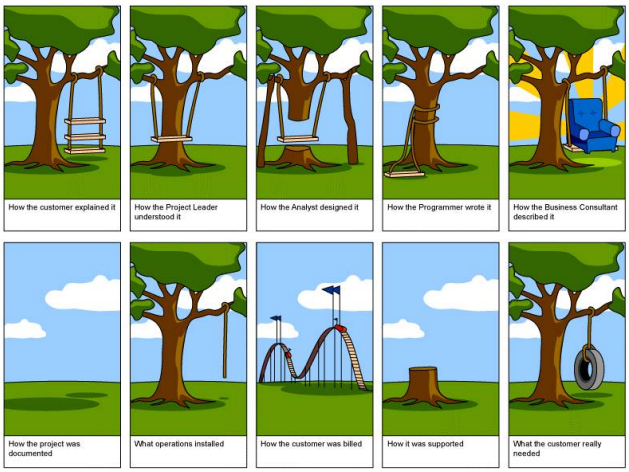
```
public enum Dwarf {
    Blick(131), Flick(112), Glick(142),
    Snick(115), Plick(108), Whick(120), Quee(99);

    int age;

    Dwarf (int a) {
        age = a;
    }

    public String toString() {
        return super.toString() + " " + age;
    }

    // fler metoder
}
```

Varje dvärg (konstant instans av klassen) definieras med ett namn och ett kontruktoranrop

```
Plick(108)    //age=108 is set in constructor
```

Skriv ut sju dvärgar i mainmetod

```
public static void main (String[] u) {
    System.out.println("All dwarfes");
    for (Dwarf d : Dwarf.values())
        System.out.print(d + "  ");
    System.out.println();
}
```



OOA Objektorienterad Analys

- Definiera VAD ett system ska göra
- Hitta objekt, klasser och relationer

Hjälpmedel

- Kravspecifikation
- Analysera kravspec och hitta
  substantiv verb relationer
  (attrib., obj., aktörer) (metoder, rel.) (associationer, arv)
- Bygg upp Data Dictionary
- Användningsfall (Use Cases)
- Brainstorming för att hitta fler klasser

Exempel på informell kravspecifikation

Some employees work by the hour. They are paid an hourly rate that is one of the fields in their employment record. They submit daily time cards that record the date and the number of hours worked. If they work more than 8 hours per day they are paid 1.5 times their normal rate for those extra hours. They are paid every Friday. Some employees are paid a flat (non-hourly) salary. They are paid on the last working day of the month. Their monthly salary is one of the fields in their employment record. Some of the salaried employees are also paid a commission based on their sales.

They submit sales receipts that record the date and amount of the sale. Their commission rate is a field in their employee record. They are paid every second Friday. Every employment record contains the name of the employee, together with any other relevant in- formation. Employees can select their method of payment. They may have their paychecks mailed to the postal address of their choice; they may have their paychecks held for pickup by the company cashier; or they can request that their paychecks be directly deposited into the bank account of their choice.

Fortsätter ....

Data Dictionary

Substantiv (några av dem)

amount – hur mycket som sålts
bank account – konto för löneinsättning
date – datum
employee – anställd med tim- eller månadslön
employment record – upgifter om anställd
hourly rate – lön per timme
name – den anställdes namn
time card – ett per dag och anställd med uppgifter om datum och antal arbetade timmar

Data Dictionary, fortsättning

Verb (några av dem)
generate payments
pay

Relationer (några av dem)
employee **has** employment record **1-1**
employment record **has** name **1-1**
salaried employee **is** employee

och så vidare ....

Utvecklingsmetodik

- Waterfall
- Code-and-fix
- Spiral
- Rapid Prototyping
- Unified Process (UP)
- Agile methods, t.ex.
  XP = eXtreme Programming
  SCRUM
- COTS

Vattenfall

- Klassisk, välkänd, vanlig (?)
- Introduerades av Royce 1970

Vattenfallsmodellens faser

- Behovsanalys
  - Kravspecifikation
    - Design
      - Implementation
        - Testning
          - Leverans
            - Drift och underhåll

Vattenfall – fördelar

- Lätt att förstå
- Förstärker goda(?) vanor:
  först design, sedan kodning
- Dokument-driven
- Lämplig för stora kända tillämpningar
  och oerfarna programmerare

Vattenfall – nackdelar

- Idealiserad, verklighetsfrånvänd
- Speglar ej den iterativa naturen
  hos programutveckling
- Orealistiskt att veta allt från början
- Programvaran kommer sent, allvarliga
  fel/problem kan upptäckas sent
- Administrationen stor och dyr,
  särskilt för små projekt

Spiralmodellen

Programutveckling är naturligt
experimentell och iterativ

1. Skriv ett program
2. Uppfyller det kraven?
3. Om NEJ goto 1, annars sluta

- Iterativ modell med riskanalys
- Varje iteration löser delproblemet med högst risk
- Varje iteration utförs "vattenfallslikt"

Flexibel och realistisk . . . men riskanalys är svårt!

Code–and–Fix

- Starta med programidé
- Skriv på tills programmet är klart
- Ingen särskild plan

Kan fungera för små "proof of concept" – projekt.

Fungerar **inte** för stora projekt !!!

Rapid Prototyping

Idé: Snabbt visa en icke-teknisk beställare vad som går att göra

- Iterera fram till slutprodukten
- Analys och design i varje steg

\+ Iterativt
\+ Kundorienterat
\+ Tydliga delresultat

– Kunden måste delta aktivt (dyrt)
– En ofullbordad prototyp kan bli slutprodukt
– Smal, kundspecifik produkt

COTS

- COTS = Commercial Off-The-Shelf software
- Bygg ihop slutprodukten från existerande programvara
- T.ex. databaser, grafikpaket, textredigerare

- Snabbt, billigt (+)
- Ger en baslösning utan finesser
- Licensproblem, licensavgifter (–)

Traditionella metoder

- är byråkratiska
- vill planera allt från början
- accepterar inte att krav ändras under tiden
- skiljer på design och konstruktion
- kan fungera för stora förutsägbara projekt
- men fungerar ofta inte bra

Lättviktiga metoder

- anpassar sig till förändring
- är iterativa
- är kodorienterade

Agile = lättviktigt, vig, flexibel, kvick

Växte fram under 90-talet

Mest kända är XP = eXtreme Programming

och SCRUM

Manifesto for Agile Software Development

Skrevs 2001 av en grupp förespråkare för "agila" metoder.

Individuals and Interactions
over processes and tools
Working Software
over comprehensive documentation
Customer Collaboration
over contract negotiation
Responding to Change
over following a plan

That is, while there is value in the items on the right
we value the items on the left more.

XP – regler (grön → svår att hålla)

- The planning game
- Small releases
- Metaphor
- Simple design
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- On-site customer
- Coding standards
- Testing
- 40-hour week

The planning game

- Görs i början av varje iteration
- Kunden beskriver önskade funktioner
- Utvecklaren uppskattar kostnaderna och tiden
- Kunden väljer funktioner
- Varje funktion detaljplaneras och arbetet fördelas

En Iteration tar 1–2 veckor

Enkel design

- Gör det enklaste som fungerar
- Gör inget mer än vad som behövs

Refactoring

- Kodförbättring utan att programmets synliga funktion ändras

Gemensamt ägande

- Alla är ansvariga för allt och får ändra (förbättra) allt

Testning

- Testa varje programdel (UNIT-testing)
- Testa hela systemet ofta
- Automatisera testningen
- Skriv testerna före själva programkoden – Testdriven programutveckling

Hjälpmedel för testning

JUnit

`http://www.junit.org`

Kritik mot XP

Extreme Programming Refactored: The Case Against XP av Matt Stephens & Doug Rosenberg

- "XP is a mixture of common sense and nonsense"
- Constant refactoring is a poor substitute for design" Planera med UML innan kod skrivs!
- "Emergent design is shortsighted and inefficient" Tillåt att man implementerar "naturliga" nästa steg

- Dogmatiska XP-företrädare
- Mycket kritik
  → mycket diskussion
- Många kritiker anser att en del av XP är bra, t.ex.
  - Refactoring
  - Enkelhet
  - Iterering
  - Testning

  men att de överdrivs av XP

SCRUM

- Iterativ process för programutveckling
- Roller
  - Scrum Master
  - Utvecklingsteam
  - Produktägare
- Sprint = iteration, 3-30 dagar

  planeras, utvärderas
- Obligatoriska möten

  sprintstart, sprintslut, dagligen
- Obligatoriska dokument

Skillnader SCRUM – XP

- SCRUM är mindre tekniskt, mer om management
  - SCRUMs ursprung är ej i mjukvarukontruktion
  - SCRUM har inget om parprogrammering eller testning
- SCRUM har längre iterationer än XP
- I SCRUM har kund inflytande endast i början av sprint
- I SCRUM jobbar teamen fritt under sprint
- SCRUM Master är en permanent roll,
  i XP växlar coach-rollen

XP: endast för erfarna programutvecklare

SCRUM: för alla ?