Programmeringsteknik och Matlab
Övning 7

Övningsgrupp 2 (Sal Q22/E32)

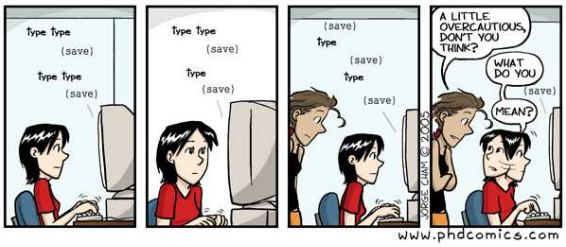
Johannes Hjorth
hjorth@nada.kth.se
Rum 4538 på plan 5 i D-huset
08 - 790 69 02

Kurshemsida:
http://www.nada.kth.se/kurser/kth/2D1312

Övningsanteckningar:
http://www.nada.kth.se/∼hjorth/teaching/prgi05

Dagens program

- Vad händer nu?
  J-uppgiften är ett självständigt projekt
- Hur skriver vi en J-spec?
  Första steget på J-uppgiften
- Var ligger ribban?
  Redovisning och betygskriterier
- Snygg kod – hur får vi det?
  Bra namnval, indentering, gruppera rader



Viktiga information

2005-10-27
Java-provet klockan 8-10 i sal röd, orange, gul och brun. Sista dagen för inlämning av J-spec.

2005-11-25
Sista dagen för redovisning av J-uppgiften för betyg högre än 3. Är uppgiften godkänd 25 november går det att plussa till högre betyg inom ett år från kursstart. Har däremot J-uppgiften inte redovisats senast detta datum kan den fortfarande redovisas med då fås bara betyg 3

2006-09-06
Sista dagen för att redovisa J-uppgift eller plussa. Intjänad bonus till provet försvinner!

Tre moment för J-uppgiften

Specifikation
Syftet med specifikationen är att ni ska tänka igenom problemet innan ni börjar programmera. Den skrivs i en java-fil och lämnas in på papper senast den 27 oktober. Lägg ner tid och tankearbete på detta eftersom ni kommer ha igen det när ni kodar.

Granskning
Före redovisning av J-uppgiften ska den granskas av en kursare. Det finns ett granskningsprotokoll på nätet att följa, länk hittar ni både på kurssidan och på min undervisningssida. Gör ni detta ordentligt kommer slutredovisningen att gå lättare.

Slutredovisning
Du bokar tid för slutredovisning på webben. Senast 25 november för betyg högre än tre. När du redovisar din J-uppgift ska även den person som granskat ditt program följa med. Medtag också det ifyllda granskningsprotokollet.

Varför ska jag skriva en J-spec?

Dagen är kommen. Du har valt din J-uppgift, laddat upp med en påse varma nybakade bullar och precis satt dig ned framför datorn.

Mobilen har du lämnat hemma så att ingenting ska kunna störa. Hela dagen har du för dig själv.

Du lägger händerna på tangentbordet och undrar stillsamt för dig själv: *hur ska jag börja?*

Då vore det trevligt att ha någon form av mall för ditt program, en första vägbeskrivning på hur du ska gå till väga.

Lyckligtvis har du det! Längst ner i ryggsäcken ligger ett par ihopknycklade papper med titeln J-specen, som du skrev i mitten av oktober.

Vad är en J-spec?

Syftet med din J-spec är att den ska vara en grov ritning för det program du snart ska skriva.

- Hur har du tänkt dela upp ditt program?
  Vilka klasser metoder, variabler och instanser behöver du?
- Hur samverkar dessa för att lösa uppgiften?
  Vilken algoritm använder du i ditt program?

Du behöver inte i detalj berätta hur varje del utför sin uppgift, bara vad det är du vill att den gör. Detta är mycket viktigt!

Ett exempel. Antag att din algoritm kräver en sorterad vektor. Du behöver då inte beskriva själva sorteringen, utan kan nöja dig med att i specen säga att du kommer att ha en metod som utför sorteringen och varifrån du vill anropa den. Hur du sedan gör sorteringen är ett senare problem!

Första steget är alltid att få en översiktlig karta över hur ditt program ska se ut innan du börjar.

Hur skriver vi en J-spec?

Det gäller att ha klart för sig vad en J-spec är och inte är, annars kanske vi gör onödigt mycket jobb.

- Det är inte en komplett beskrivning av varje liten detalj i ditt program.
- Däremot är din J-spec en grov mall för hur programmet kommer att se ut.

Vi tar ett exempel för att se vad det här innebär...

Luffarschack, ett exempel på J-spec

Vi är tokiga i luffarschack! Valet av J-uppgift faller sig därför mycket enkelt för oss...

Men hur börjar vi? Vad ska programmet göra?

Vi börjar med att göra en ungefärlig lista över vad som bör ske när programmet körs:

- Visa en spelplan
- Fråga efter spelarens drag
- Utföra datorns drag
- Kontrollera om spelet är slut

Utan att vi har ansträngt oss har vi här fått en grov algoritm för vårt spel.

Hur kan vi dela upp uppgifterna?

Vad behöver vi för att kunna spela?

- En spelplan
- En människospelare
- En datorspelare

Om vi börjar med spelplanen. Antag att vi har en klass som representerar den. Vad vill vi då att den klassen ska kunna göra?

- Hålla reda på ringarna och kryssen på spelplanen
- Kontrollera att dragen som spelarna gör är giltiga
- Se om någon lyckats få fem i rad

Om vi också tar Människospelare som en klass. Vad ska den då göra? Ett förslag är följande:

- Fråga spelaren efter namn
- Fråga spelaren vilket drag han eller hon vill göra
- Verifiera med spelplanen att draget är giltigt

Slutligen datorspelaren, dess uppgift är bara att komma på ett smart drag. Hur gör den det – ja det är ett senare problem. Specen är bara en mall.

Vad behöver klasserna veta?

Vad är det våra klasser behöver hålla reda på för att kunna lösa sin uppgift? Vi börjar med det enklaste.

Spelplanen behöver hålla reda på vilka drag som gjorts. Hur lagrar vi detta på ett smart sätt?
Ett sätt att göra detta är i en två-dimensionell array av datatypen `int`.

Människospelaren behöver hålla reda på namnet på personen som spelar. Lämpligen som en sträng.

Datorspelaren är lite lurigare och lämnas som övning till den intresserade.

Hur skriver vi mallen i java-kod?

Själva tanken med specen är att vi vill få ner ett förslag på klassens struktur på papper.

```
public class Spelplan {

    private int spelplan[][]; // spelplanen 0 = tom, 1 = ring, 2 = kryss
    private int rader, kolumner;

    Spelplan(int rader, int kolumner) { // konstruktorn, behöver veta storlek

    }

    public void skrivUt() { // Skriver ut spelplanen
        System.out.println("* Tjusig spelplan *");
    }

    public boolean giltigtDrag(int x, int y) {
        return true; // måste returnera något för att kunna kompilera
    }

    public boolean görDrag(int x, int y, boolean kryss) {
        return true; // det var ett giltigt drag
    }

    public int vinnare() {
        return 1; // 0 = ej färdigt, 1 = ring vann, 2 = kryss vann
    }
}
```

På samma sätt skriver vi sedan en mall för datorspelaren och människospelaren.

Gör java-filen kompilerbar

Notera att metoderna lämnas tomma. Vi säger bara vad de heter och vad de gör, inte hur de ska utföra sina respektive uppgifter.

Vi sparar koden på förra sidan i en ny fil som vi kallar `Spelplan.java`

Den kan nu kompileras för att se om vi skrivit några uppenbara fel i koden:

```
>javac Spelplan.java
```

Hade vi här upptäckt några fel så vet vi direkt var vi ska leta. Det går då fortare att korrigera misstagen man gör.

Fördelen med att göra er J-spec kompilerbar, är att ni sedan kan bygga vidare på den när ni ska börja programmera.

Man behöver inte skriva om allt från början utan kan utgå från strukturen i specen och sedan fylla de tomma metoderna med kod.

Koppla samman delarna till ett spel

```
public class Luffarschack {

    private static final int oavgjort = 0; // en konstant

    private Spelplan spelplan; // instansvariabel med spelplanen
    private Datorspelare datorspelare;
    private Mänskligspelare människospelare;

    public static void main(String args[]) {

        Luffarschack luff = new Luffarschack();

        while(luff.spela() == oavgjort); // spela tills de är slut
    }

    Luffarschack() { // konstruktorn skapar de objekt spelet använder
        spelplan = new Spelplan(20,20);

        // Spelarna behöver kunna se spelplanen, eller hur?
        datorspelare = new Datorspelare(spelplan);
        människospelare = new Mänskligspelare(spelplan);
    }

    public int spela() { // returnerar 0 om ej färdig
        spelplan.skrivUt();
        människospelare.görDrag();
        datorspelare.görDrag();

        return spelplan.vinnare();
    }
}
```

Notera hur indentering av koden och några tomma rader gör programmet mer läsbar.

Kompilera och testkör

Vi kan kompilera och testköra vår J-spec. Utskriften som kommer ut är dock inte speciellt rolig.

```
>javac Luffarschack.java
>java Luffarschack
* Tjusig spelplan *
>
```

Ett sätt att liva upp det lite är att lägga till utskrifter i alla metoder som inte är färdiga:

```
System.out.println("Här körs datorns görDrag-metod");
```

Då är det lättare att följa flödet i programmet och se att det blir som vi vill.

Betygskriterier

För att bli godkänd på kursen måste man klara av de tre momenten: labbar, prov och J-uppgiften.

- **E (3)** Godkänd J-redovisning med en riktig minnesbild av uppgift med fördefinierad spec/skelettprogram.
- **D (3)** Godkänd J-redovisning med en riktig minnesbild av uppgift med egenhändigt skriven spec.

För betyg högre än D krävs att grunduppgiften redovisas före kursomgångens slut.

- **C (4)** Kraven för D samt ett perfekt program med korrekt hantering av felaktig inmatning, dvs inga anmärkningar i protokollet (väl uppdelat, ingen koduppprepning, vettigt dokumenterat, dock utan krav på javadoc mm). Alternativt en uppgift med betyg från B på grunduppgiften och godkänd redovisning (dvs max tre påpekanden).
- **B (4)** Uppgift eller extrauppgift med betyg B som är perfekt, dvs inga anmärkningar i protokollet och med korrekt hantering av felaktig inmatning. Observera att kraven på ett perfekt program gäller hela programmet, inklusive ev extrauppgift.
- **A (5)** Kraven för C samt en extrauppgift med grafik eller avancerad algoritm. Observera att kraven på ett perfekt program gäller hela programmet, inklusive extrauppgiften.

Checklista

- Vad behöver vi hålla reda på för att kunna lösa vårt problem? Gör en lista. Variabler, datatyp?
- Finns det någon naturlig uppdelning av problemet, vilka klasser kommer vi att behöva?
- Tänk igenom flödet i programmet. Kan vi dela upp problemet i olika mindre delproblem?
- Vad behöver varje delproblem veta?
- Skapa metoder som löser delproblemen. Vilken klass bör metoden ligga i, vilka variabler behöver den känna till?
- Ge metoder namn som beskriver vad de gör, inte hur de gör det. Detta gör koden mer lättläst.
- Gör alla instansvariabler `private`.
- Räkna till tio varje gång du skriver `static` någonstans, vill du verkligen ha en *klassmetod*?