TDDE10
725G90

Objektorienterad programmering i Java
Föreläsning 8

Emma Enocksson Svensson & Magnus Nielsen

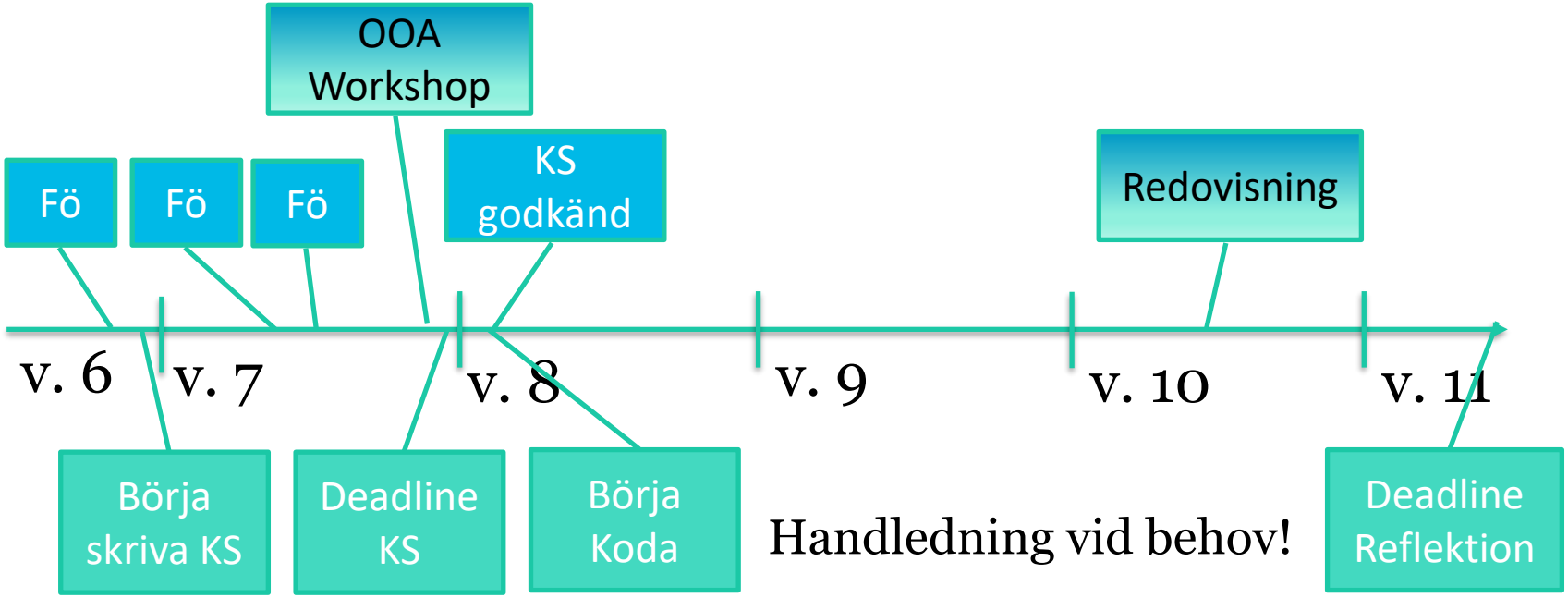
LINKÖPINGS UNIVERSITET

Dagens föreläsning

- Projektet
  - Omfattning
  - Upplägg
  - Kravspecifikation
- Tips & Tricks!

Projektet

- **Tidsplan**



- 2024-02-16 kl 15.00 : Deadline Kravspecifikation
- 2024-03-06 kl 12.00 : Redovisning
- 2024-03-11 kl 17.00 : Reflektion + Klassdiagram inskickad

Projekt

- **Omfattning**
  - Ett arkadspel skall göras!
  - 1 Spelare mot spelet
  - Grafiskt gränssnitt som styrs med tangentbord och mus
  - Flera banor, fiender, powerups o.s.v.
  - Meny, med bl.a. highscore (d.v.s. någon typ av poäng behövs)

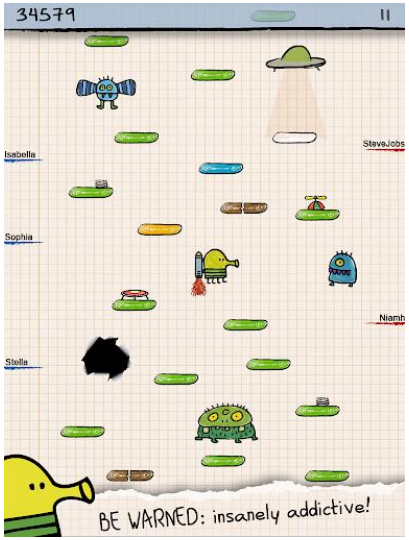
Projekt

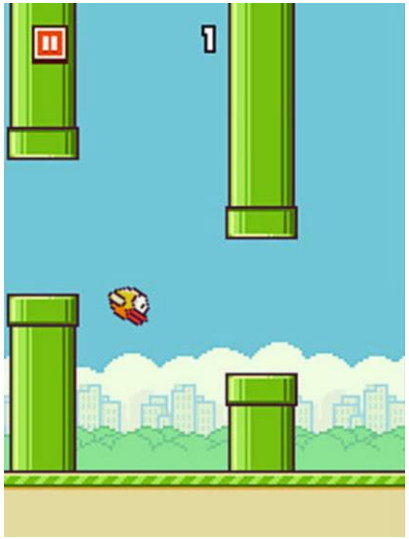
- **Resurser**
  - Grupper om 2 (ni får välja om ni vill byta partner eller ej)
  - Ny anmälan i webreg till handledare
    - (OBS! Vi kan komma att balansera mellan assistenter så att deras arbetsbelastning blir jämnare)
  - Ca 3 veckor kalendertid
  - 3.5 föreläsningar (Krav & Tips, Blandat, Versionshantering)
  - En lektion (OOA-workshop)
  - GitLab

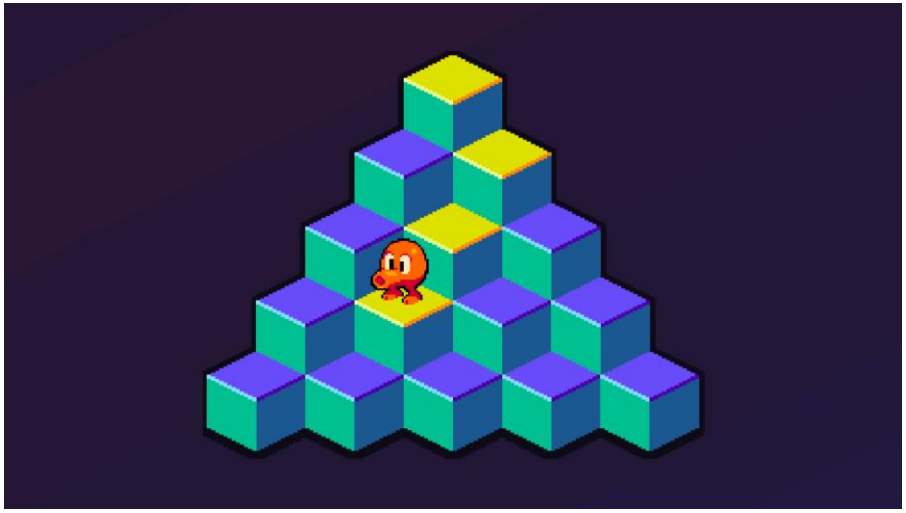
Projektet är i stort sett helt självständigt arbete

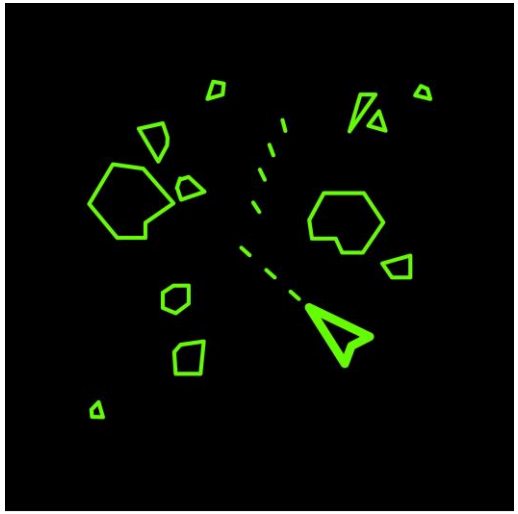
Projekt

Exempel



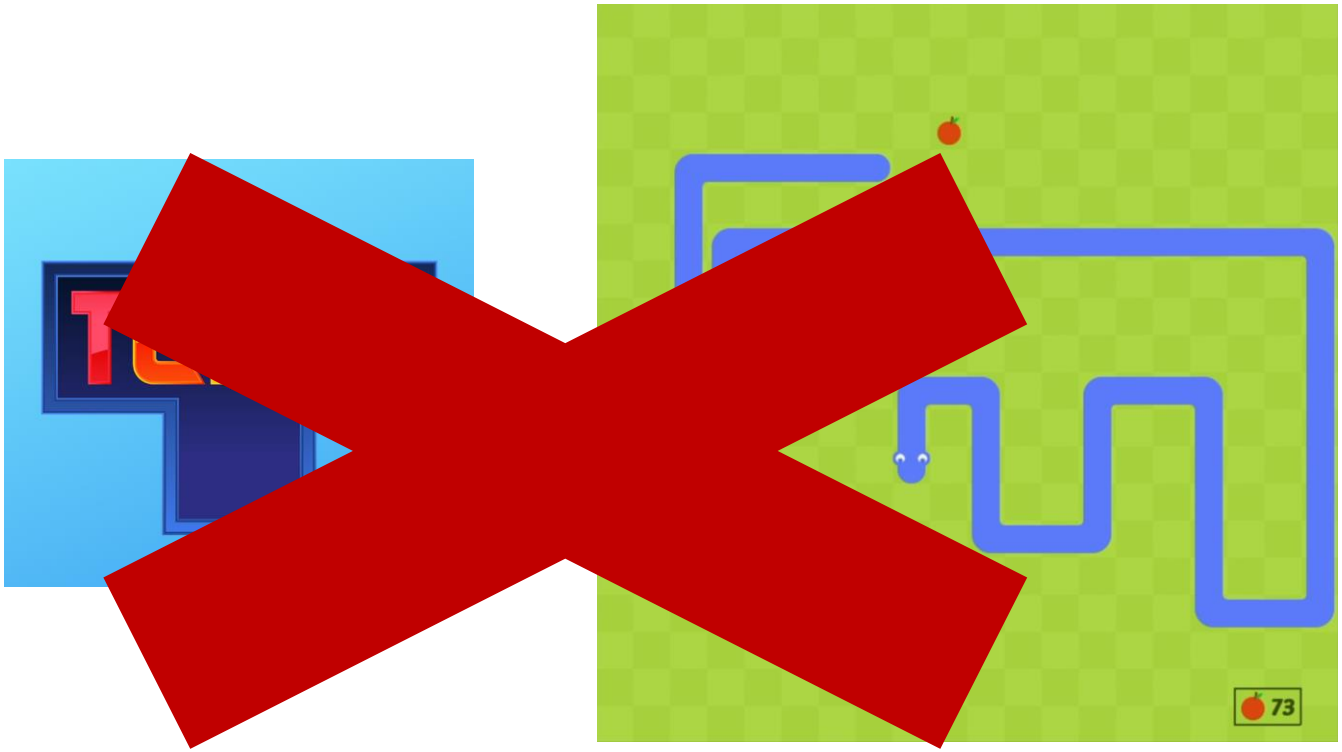






Ej tillåtna projekt

Exempel



**Vi reserverar oss för att inte tillåta fler än dessa spel när vi läser er kravspec*

Projekt

- **Övriga krav (icke-funktionella)**
  - Skall vara skrivet i Java
  - Får använda annat grafiskt lib än FX, men då kan man inte förvänta sig hjälp med det
  - Skall versionhanteras med git/GitLab
  - Handledaren skall bjudas in till git-projektet
  - Åtminstone 2 moduler (paket)
    - Lämpligt att dela på grafik och logik
  - Javadoc skall finnas på klasser (endast kort beskrivning av vad klassen representerar krävs)

Projekt

- **Redovisning**
  - Godkänd kravspec
  - Godkänd muntlig demo vid redovisningstillfället
  - Kraven uppfyllda
  - Koden ok
  - Godkänt reflektionsdokument
    - 1 per grupp
    - 2-3 A4 om **ert lärande** under projektet
    - Översiktligt klassdiagram

Kravspecifikation

- **Vad skall ingå**
  - Er *vision*, d.v.s. en löpande text om hur spelet skall vara (Ca 1 A4 text)
  - En eller fler bilder (skisser) över hur spelet ser ut när man spelar, med förklarande texter. Helt ok att rita för hand och infoga
  - En tabell över konkreta, funktionella **krav**. Ca 15 – 25 st är rimligt.
  - Skickas som **PDF** till handledaren senast **kl. 15.00 fredagen den 16:e Februari**
  - Handledaren ger er svar senast på måndagen. När kravspec:en är godkänd så kan ni börja koda!

Kravspecifikation

Fokus för detta projekt

- **Hur är en kravspec för mjukvara?**
  - Beskriver funktionaliteten. *Vad programmet skall göra*
  - Beskriver gränssnitt(en). *Hur programmet interagerar med omvärlden (människor såväl som andra system)*
  - ~~Beskriver andra attribut, t.ex.~~
    - ~~Portabilitet, hållbarhet, prestanda, säkerhet, o.s.v.~~
  - ~~Anda begräsningar för systemet, t.ex.~~
    - ~~Implementationsspråk, resursbegränsing, o.s.v.~~
  - Går inte in på design eller implementationsdetaljer!

IEEE Std 830-1998:
Recommended Practice for System Requirements Specifications

Kravspecifikation

- **Vilka krav behövs?**
  - Kraven arbetas fram tillsammans med kund och andra intressenter
  - Ofta används use-cases och scenarion även här, för att kunna identifiera systemets viktiga delar

  - I detta projekt agerar ni både kund och utvecklare
  - Ni får alltså själva komma fram till vilka krav som behövs!
  - **Tips:** Använd er vision och era skisser
  - Se även mini-kraven på kurshemsidan
  - Vi (handledare & föreläsare) finns med som yttre intressenter

Kravspecifikation

- **Tips (från tidigare år)**
  - Skissa er vision först
  - Diskutera mini-kraven och hur de passar er vision
  - Utför en OOA (mer om det på nästa föreläsning)
  - Skapa ett UML, ett litet Designdokument eller något att utgå ifrån
  - Skapa en krav-lista

Kravspecifikation

- **Hur skriver man ett krav?**
- Det finns vissa karaktäristiska som är viktiga för krav. Ett krav skall vara:
  - **Korrekt** : Om kravet inte skall gälla för programmet så är det inte korrekt
  - **Konsistent** : Ett krav får inte motsäga ett annat krav, eller andra delar av kravspec:en (t.ex. visionen eller skissen)
  - **Icke tvetydigt** : Ett krav får inte ha flera tolkningar. Termer kan behöva definieras i en ordlista
  - **Rankat** : Vissa krav är viktigare än andra. Alla krav bör ha något mått så att man vet vad som bör prioriteras
  - **Verifierbart** : Det måste gå att avgöra om kravet är uppfyllt
  - **Spårbart** : Vi måste kunna referera till ett krav. Själva texten i ett krav kan ju ändras. Görs lättast med nummer
  - **Icke redundant** : Om ett krav överlappar med ett annat kan detta leda till fel senare …

IEEE Std 830-1998:
Recommended Practice for System Requirements Specifications

Kravspecifikation

- **Exempel**

Det skall finnas åtta olika former i spelet.
Spelaren skall kunna rotera en form genom att trycka på en tangent.
Spelet skall visa upp spelarens nuvarande poäng.
Spelet skall visa vilken av de sju formerna som kommer härnäst.
Formerna skall dyka upp i spelets underkant och röra sig nedåt.
När en rad är komplett så skall den försvinna och spelaren får poäng för den.
Former kan roteras med mellanslag så länge de inte har landat.
Spelets grafiska design skall vara snygg och användarvänlig.

Ev. redundant

Ej konsistent

Tvetydigt
Vad är ”komplett”?
Vad är ”en rad”?

Ej spårbara

Ej verifierbart
Vad är ”snyggt?”
Vad är ”användarvänligt”?

Ej korrekt

Ej rankade

IEEE Std 830-1998:
Recommended Practice for System Requirements Specifications

Kravspecifikation

- **Exempel**

ID	Beskrivning	Prio
1.	Det skall finnas **sju** olika former i spelet.	1
2.	Spelaren skall kunna rotera en form genom att trycka på en tangent.	1
3.	Spelet skall visa upp spelarens nuvarande poäng.	1
4.	Spelet skall visa vilken av de ~~sju~~ formerna som kommer härnäst.	1
5.	Formerna skall dyka upp i spelets **överkant** och röra sig nedåt.	1
6.	När en rad är komplett så skall den försvinna och spelaren får poäng för den. *Se definitionerna för ”rad” och ”komplett” i ordlistan.*	1
7.	Former kan roteras ~~med mellanslag~~ så länge de inte har landat.	1
8.	**Varje form skall ha en egen färg.**	2

IEEE Std 830-1998:
Recommended Practice for System Requirements Specifications

Kravspecifikation

- **Exempel**

ID	Beskrivning	Prio
1.	Det skall finnas sju olika former i spelet.	1
2.	Spelaren skall kunna rotera en form genom att trycka på en tangent.	1
3.	Spelet skall visa upp spelarens nuvarande poäng.	1
4.	Spelet skall visa vilken av de formerna som kommer härnäst.	1
5.	Formerna skall dyka upp i spelets överkant och röra sig nedåt.	1
6.	När en rad är komplett så skall den försvinna och spelaren får poäng för den. *Se definitionerna för ”rad” och ”komplett” i ordlistan.*	1
7.	Former kan roteras så länge de inte har landat.	1
8.	Varje form skall ha en egen färg.	2

IEEE Std 830-1998:
Recommended Practice for System Requirements Specifications

Kravspecifikation

- **Övriga tips**
  - Låt åtminstone 3 av era krav vara lägre prioriterade så att ni har några extra saker att göra om ni blir klara tidigt. Om ni inte hinner med dessa så är det inte hela världen
  - Ni behöver inte fler än 3 prio-nivåer där nivå 1 är viktigast
  - Välj något välkänt spel och gör er egen variant av det
  - Ni hinner troligtvis inte att både designa ett nytt spel och implementera det

Tips-Time!

Tips-time!

- **Ritprogrammet är ju nästan ett spel…**
  - Ni kan säkert återanvända en del kod, eller åtminstone idéer från det
- **Kolla på koden från denna föreläsning och förra!**
  - Kommer finnas fler små exempel på tipsen nedan
- **Även här: jobba objektorienterat!**
  - Ni kommer behöva ändra mycket i koden medan ni jobbar
  - Använd det ni lärt er!
  - D.v.s. klasshirarkier, polymorfi, abstrakta klasser, inkapsling

Spel-Loop

- De allra flesta spel har någon form av spel-loop
- Generell form:

```
while (speletEjSlut) {
  hanteraIndata();
  uppdateraModell();
  ritaUtAllt();

 väntaEnKortStund(20ms);
}
```

t.ex mus, tangentbord

All ”logik” uppdateras

Uppdateringarna visas för spelaren

Spel-Loop

- Generell form:

```
while (speletEjSlut) {
  hanteraIndata();
  uppdateraModell();
  ritaUtAllt();

  väntaEnKortStund(20ms);
}
```

Med FX behövs eventuellt inte denna, då indata kan hanteras av lyssnare och händelser.

Modellen går igenom alla sina objekt och talar om för dem att uppdatera sig själva.
Ex.

```
For (ModelObject m: modelObjects) {
  m.updateYourself(this);
}
```

T.ex. repaint() eller paintImmediatly() på GUIt

T.ex.
Thread.sleep(20);

Spel-Loop

- Thread.sleep() kan variera ett par ms.
- Uppdatering av modellen kan också ta variabel tid
- Egentligen bör man mäta hur lång tid varje del i spel-loopen tar
  - Kan göras med System.currentTimeMillis()
- Därefter räknar man ut hur lång tid man skall vänta
- Uppdatering av modellen bör då bero av tiden som gått sedan föregående uppdatering
- Eller….

Spel-Loop

- Finns en inbyggd AnimationTimer-klass i Java man kan använda sig av!

```
final double targetFps = 50.0;
final double nanoPerUpdate = 1000000000.0 / targetFps;
new AnimationTimer() {
    long lastUpdate = 0;
    public void handle(long now) {
        if ((now - lastUpdate) > nanoPerUpdate) {
            myModel.update();
            frame.repaint();
            lastUpdate = now;
        }
    }
}.start();
```

Sätter vad vi ska sikta på med frames per second

Beräknar frekvensen i nanosekunder

Hanterar spelets flöde

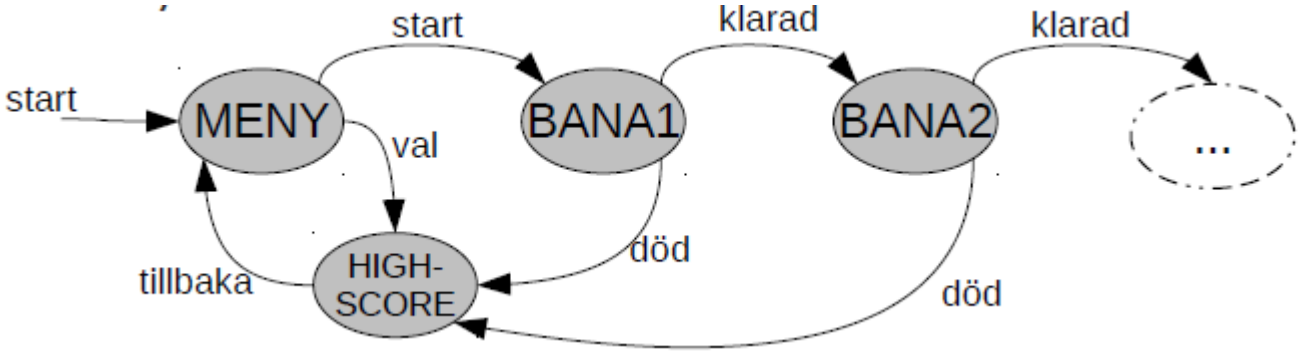
Uppdaterar spelet genom att uppdatera modellen och sedan rita om den.

Den kommer försöka göra detta ungefär 60 gånger per sekund, eller så nära det som möjligt i tid.

Kan variera stort baserat på system eller hur mycket det är att göra vid uppdatering så if-satsen kan hjälpa till att hålla det jämnt vid vår mål-FPS.

Tillstånd (eng. state)

- Er spelmodell behöver troligtvis hålla reda på spelets ***tillstånd***. D.v.s. vilken bana man är på, eller om man är i menyn o.s.v.



- Detta kan göras på olika sätt, men löses bäst med arv och polymorfi
- Tänk er en klass State där de olika tillstånden ärver från dessa. I er modell har man sedan ett system som byter mellan dessa tillstånd beroende på exempelvis ny bana.
- Smidigt att kombinera ihop med Scenes

Kort om trådar…

- Tänk på att inte ”blockera” trådar. Låt händelser *löpa ut*
- Exempel:

```
setOnMouseClicked(event -> {
  while (!gameOver) {
    // do things
  }
});
```

Här ”blockeras” händelsetråden och andra trådar får inte en chans att göra sitt. Medför att programmet verkar ha hängt sig…

Kort om trådar…

- Tänk på att inte ”blockera” trådar. Låt händelser *löpa ut*
- Exempel (denna gång i utritningstråden) :

```
protected void paintComponent(GraphicsContext gc) {
  while (!weAreDoneHere) {
    // do things
  }
});
```

Här ”blockeras” utritningstråden och andra trådar får inte en chans att göra sitt. Medför att programmet verkar ha hängt sig…

- Alltså : lägg spel-loopen i huvudprogrammet
- Låt en klass (kanske modellen?) hålla reda på vilket tillstånd spelet är i

Buffra bilder

- Ni kommer säkert vilja ha bilder, s.k. ”sprites” i ert spel.
- Enklast är det nog att låta objekten själva rita ut dessa bilder:

```
public void paintYourself(GraphicsContext gc) {
  Image cover = new Image(new
                FileInputStream("/home/emmen85/Downloads/cover.png"));
  gc.drawImage(cover, x, y);
};
```

- Men vad händer när vi ritar ut många objekt?
  - Alla dessa skall läsa filen varje gång spelet skall ritas ut. Även en liten bild kan då ta tid…

Buffra bilder

- Lösning 1: Spara bilden som en klassvariabel

```
private static Image image = null;

private static Image getImage() {
   if (image == null) {
     image = new Image(new
                    FileInputStream("/home/emmsv874/Downloads/cover.png"));
  }
  return image;
}
public void paintYourself(GraphicsContext2D gc) {
   gc.drawImage(getImage(), x, y);
};
```

Buffra bilder

- Lösning 2: Spara alla bilder i en separat manager-klass
  - Denna klassen ansvarar för att bilder läses in en gång
  - Lämpligtvis sparar klassen undan dem i en map (dictionary, inte mapp)

```
public void paintYourself(GraphicsContext gc) {
  gc.drawImage(SpriteManager.getImage(”cover.png”), x, y);
};
```

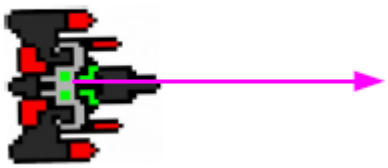
Buffra bilder

- Andra tips om ”sprites”
  - GraphicsContext-objekt kan rita ut bara en del av en bild
  - Kan vara praktiskt att använda ”sprite maps”

  - Animationer kan vara en sprite map där man ”tickar” fram mellan bilderna
  - GraphicsContext kan hantera transparens i bilder

Kollisioner

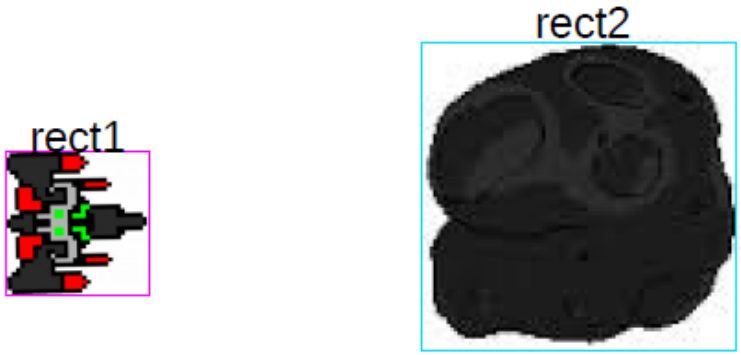
- Ni kommer troligtvis behöva en del klasser för att sköta geometri i ert program
- T.ex. hur vet vi om skeppet kolliderat med asteroiden?



- Finns många bra klasser i javafx.geometry T.ex:
  - Rectangle2D, för att representera en rektangel
  - Point2D, en representation av en punkt i ett plan
  - BoundingBox, ett objekt med en bredd och höjd

Kollisioner

- **Möjlig lösning :** Se objekten som rektanglar och kolla om dessa överlappar




```
Rectange2D rect1 = new Rectange2D(ship.getX(), ship.getY(),
                                  ship.getWidth(), ship.getHeight());

Rectange2D rect2 = new Rectange2D(rock.getX(), rock.getY(),
                                  rock.getWidth(), rock.getHeight());

if (rect1.intersects(rect2)) {
  // objekten har kolliderat!
}
```

GraphicsContext

- I paintComponent-metoden kan man utnyttja inbyggda metoder hos GraphicsContext

- GraphicsContext har metoder för att:
  - Rotera object
  - Flytta kordinatsystem (eng. translate)
  - Förstora/Förminska (eng. scale)

Concurrent Modification

- Antag att vi i modellen (GameModel) har följande någonstans i en metod:

```
for (GameObject obj : allObjects) {
  obj.doYourStuff(this);
}
```

- Och att vi även har denna metod i GameModel:

```
public List<GameObject> getAllObjects() {
  return allObjects;
}
```

- Vad händer om vi i doYourStuff gör detta?

```
gameModel.getAllObjects().remove(...)
```

Concurrent Modification

- Vi ändrar ju på en lista medan vi itererar igenom den…
- Vägar runt detta:
  - Kopiera över alla objekt till en annan lista, iterera över den men ta bort i orginalet
  - ”Markera” objekt för borttagning
  - Samla objekt som skall tas bort i en ”skräplista”, och ta bort dem efter
  - I både de senaste två fallen, ta bort elementen efter for-each-loopen är klar

emma.enocksson@liu.se
magnus.nielsen@liu.se

www.liu.se

LINKÖPINGS UNIVERSITET