# Kapittel 8:
# Programutvikling

***Redigert av:***

Khalid Azim Mughal (`khalid@ii.uib.no`)

***Kilde:***

*Java som første programmeringsspråk (3. utgave)*

Khalid Azim Mughal, Torill Hamre, Rolf W. Rasmussen
Cappelen Akademisk Forlag, 2006.
ISBN: 82-02-24554-0
`http://www.ii.uib.no/~khalid/jfps3u/`
*(NB! Boken dekker opptil Java 6, men notatene er oppdatert til Java 7.)*

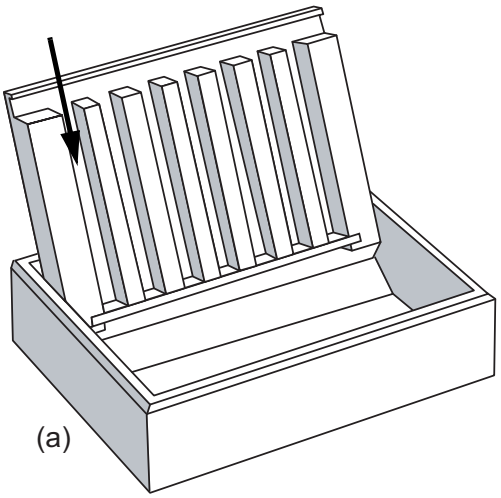


## Emneoversikt

- Formulering av problemstilling
- Enkel testing med assert
  - Implementasjon som oppfyller test
  - Test endring av objekt tilstand
  - Kjøring av test før implementering av funksjonalitet
  - Implementasjon for endring av tilstand
  - Synliggjøre behovet for en bedre implementasjon med tester
  - Implementere ny funksjonalitet forventet av testene
- Meldinger ved feilende tester
  - Bedre meldinger ved feilende tester
- Testrammeverket JUnit
  - Hjelpemetoder for testing i JUnit
  - Feilmeldinger fra JUnit
- Refaktorering av programkode
  - Før refaktorering av programkode
  - Etter refaktorering av programkode
- Kontrakter
  - To implementasjoner av en kontrakt
- Klassebibliotek
  - Import av klassenavn
  - Import av statiske medlemsnavn
  - Utforming bibliotek og rammeverk
  - Innkapsling
  - Adgangsmodifikatorer for pakkemedlemmer
  - Adgangsmodifikatorer for klassemedlemmer
  - Programmerbart grensesnitt

# Formulering av problemstilling

Strategispillet fire-på-rad

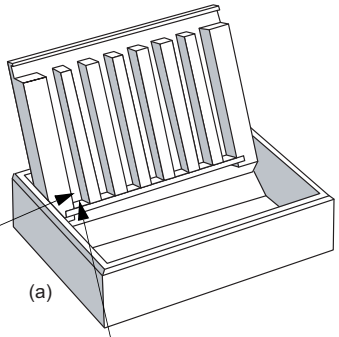



En brikke blir sluppet ned i første kolonne i et tomt spillebrett.
Hvor havner brikken?



# Enkel testing med assert

```
public class TestSpillebrett {
  public static void main(String[] args) {
    testTomRute();
  }

  public static void testTomRute() {
    Spillebrett brett = new Spillebrett();
    // Nederste rad har indeks 0.
    int rad = 0;
    // Kolonne lengst til venstre har også indeks 0
    int kolonne = 0;

    // Punktum ('.') nedfeller tom rute.
    assert brett.brikkeVed(kolonne, rad) == '.';
  }
}
```



## Implementasjon som oppfyller test

Test:

```
assert brett.brikkeVed(kolonne, rad) == '.';
```

Implementasjon:

```
public class Spillebrett {
  char brikkeVed(int kolonne, int rad) {
    return '.';
  }
}
```



## Test endring av objekt tilstand

Test:

```
public static void testSlippBrikke() {
  Spillebrett brett = new Spillebrett();
  int kolonne = 0;
  brett.slippBrikkeNedKolonne('X', kolonne);
  // Brikken skal nå ligge i kolonnen lengst til venstre på nederste rad:
  assert brett.brikkeVed(0, 0) == 'X';
}
```

---

Kompileringsfeil:

```
TestSpillebrett.java:22: cannot find symbol
symbol  : method slippBrikkeNedKolonne(char,int)
location: class Spillebrett
    brett.slippBrikkeNedKolonne('X', kolonne);
         ^
1 error
1 warning
```

## Kjøring av test før implementering av funksjonalitet

Ny implementasjon som kompilerer rett:

```
public class Spillebrett {
  char brikkeVed(int kolonne, int rad) {
    return '.';
  }

  void slippBrikkeNedKolonne(char brikke, int kolonne) {}
}
```

tom implementasjon

Feil ved kjøring av test:

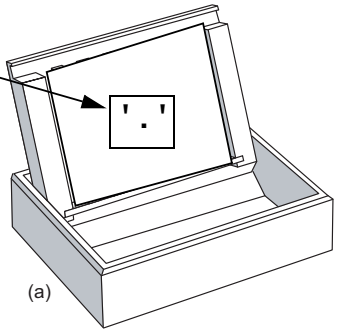
```
> java -enableassertions TestSpillebrett
Exception in thread "main" java.lang.AssertionError
at TestSpillebrett.testSlippBrikke(TestSpillebrett.java:24)
at TestSpillebrett.main(TestSpillebrett.java:5)
>
```



## Implementasjon for endring av tilstand

fungerer kun for en rute

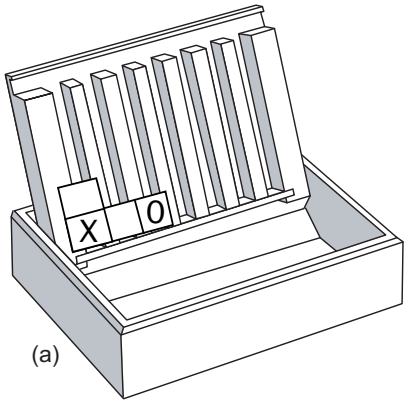
```
public class Spillebrett {
  char ruteInnhold = '.';

  char brikkeVed(int kolonne, int rad) {
    return ruteInnhold;
  }

  void slippBrikkeNedKolonne(char brikke, int kolonne) {
    ruteInnhold = brikke;
  }
}
```




Test er vellykket fordi kun tilstanden til en rute blir testet.

## Synliggjøre behovet for en bedre implementasjon med tester

- Inneværende implementasjon har begrensninger.
- Inneværende tester belyser ikke disse begrensningene.
- Lag en ny test som illustrerer behovet for ny funksjonalitet i implementasjonen.




```
public static void testSlippBrikke() {
  Spillebrett brett = new Spillebrett();
  brett.slippBrikkeNedKolonne('X', 0);
  brett.slippBrikkeNedKolonne('O', 2);
  assert brett.brikkeVed(0, 0) == 'X';
  assert brett.brikkeVed(1, 0) == Spillebrett.TOM_RUTE;
  assert brett.brikkeVed(2, 0) == 'O';
  assert brett.brikkeVed(0, 1) == Spillebrett.TOM_RUTE;
}
```



## Implementere ny funksjonalitet forventet av testene

```
public class Spillebrett {
  public static final char TOM_RUTE = '\u0000';

  char[][] ruteInnhold = new char[7][6];

  Spillebrett() {
    assert ruteInnhold[0][0] == TOM_RUTE;
  }

  char brikkeVed(int kolonne, int rad) {
    return ruteInnhold[kolonne][rad];
  }

  void slippBrikkeNedKolonne(char brikke, int kolonne) {
    int nedersteLedigeRad = 0;
    ruteInnhold[kolonne][nedersteLedigeRad] = brikke;
  }
}
```

# Meldinger ved feilende tester

Påstand med vanlig assert nøkkelord:

Linje 48:
```
assert brett.brikkeVed(3, 0) == 'X';
```

Feilmelding forteller linjenummer, men ikke hvilken brikkeverdi som lå ved (3, 0):

```
Exception in thread "main" java.lang.AssertionError
    at TestSpillebrett.testStabling(TestSpillebrett.java:48)
    at TestSpillebrett.main(TestSpillebrett.java:6)
```



# Bedre meldinger ved feilende tester

Hjelpefunksjon:

Linje 40:
```
static void påståLikhet(String melding, char forventet, char faktisk) {
  if (forventet != faktisk) {
    System.err.printf("Likhetsfeil: %s: Forventet <%s>, men fikk <%s>\n",
        melding, forventet, faktisk);
  }
  assert forventet == faktisk;
}
```

Påstand med hjelpefunksjon:

Linje 56:
```
påståLikhet("Første sluppet", 'X', brett.brikkeVed(3, 0));
```

Feilmelding med nyttig informasjon:

```
Likhetsfeil: Første sluppet: Forventet <X>, men fikk <O>
Exception in thread "main" java.lang.AssertionError
at TestSpillebrett.påståLikhet(TestSpillebrett.java:40)
at TestSpillebrett.testStabling(TestSpillebrett.java:56)
at TestSpillebrett.main(TestSpillebrett.java:6)
```

## Testrammeverket JUnit

- JUnit er et av de mest populære testrammeverkene for Java.
- Har en samling av hjelpemetoder for testing av programmer.

- Last ned et zip-arkiv med nyeste versjon av JUnit ved å følge «Download»-lenken på nettsiden http://www.junit.org/.
- Pakk ut zip-arkivet til en egnet katalogsti.
- Legg katalogsti/junit.jar til miljøvariabelen CLASSPATH.

- Sjekk først om JUnit er skikkelig installert:

```
> java junit.textui.TestRunner -v
JUnit 4...
>
```



## Hjelpemetoder for testing i JUnit junit.framework.Assert

*Påstå likhet:*

```
static void assertEquals(datatype forventet, datatype faktisk)

static void assertEquals(String beskjed, datatype forventet, datatype faktisk)
```

*Påstå sannhet:*

```
static void assertTrue(String beskjed, bool betingelse)
```

*Påstå usannhet:*

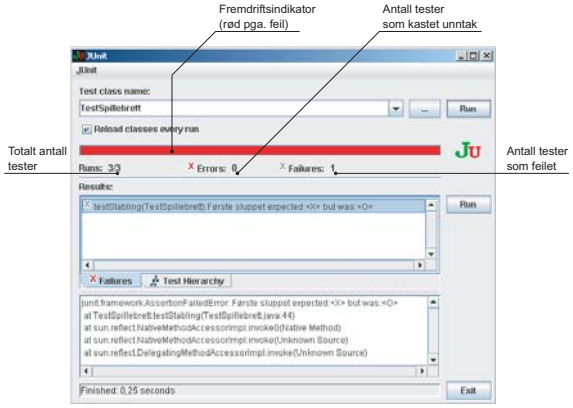
```
static void assertFalse(String beskjed, bool betingelse)
```

*Rapporter feil:*

```
static void fail(String beskjed)
```

# Feilmeldinger fra JUnit

```
> java -enableassertions junit.textui.TestRunner TestSpillebrett
...F
Time: 0,015
There was 1 failure:
1) testStabling(TestSpillebrett)junit.framework.AssertionFailedError: Første sluppet
expected:<X> but was:<O>
at TestSpillebrett.testStabling(TestSpillebrett.java:44)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

FAILURES!!!
Tests run: 3,  Failures: 1,  Errors: 0
```






# Før refaktorering av programkode

```
static String spillebrettRadSomTekst(Spillebrett brett, int rad) {
  final int ANTALL_KOLONNER = 7;
  final int SISTE_KOLONNE = ANTALL_KOLONNER - 1;
  String radTekst = "|";
  for (int kolonne = 0; kolonne < SISTE_KOLONNE; ++kolonne) {
    char tilstand = brett.brikkeVed(kolonne, rad);
    if (tilstand == Spillebrett.TOM_RUTE) {
      radTekst += ".";
    } else {
      radTekst += tilstand;
    }
    radTekst += " ";
  }
  char tilstand = brett.brikkeVed(SISTE_KOLONNE, rad);
  if (tilstand == Spillebrett.TOM_RUTE) {
    radTekst += ".";
  } else {
    radTekst += tilstand;
  }
  return radTekst + "|";
}
```

duplisert programkode

# Etter refaktorering av programkode

```
static String spillebrettRadSomTekst(Spillebrett brett, int rad) {
  final int ANTALL_KOLONNER = 7;
  final int SISTE_KOLONNE = ANTALL_KOLONNER - 1;
  String radTekst = "|";
  for (int kolonne = 0; kolonne < SISTE_KOLONNE; ++kolonne)
    radTekst += spillebrettRuteSomTekst(brett, kolonne, rad) + " ";
  return radTekst + spillebrettRuteSomTekst(brett, SISTE_KOLONNE, rad) + "|";
}
```

```
static char spillebrettRuteSomTekst(Spillebrett brett, int kolonne, int rad) {
  char tilstand = brett.brikkeVed(kolonne, rad);
  if (tilstand == Spillebrett.TOM_RUTE)
    return '.';
  return tilstand;
}
```



# Kontrakter

- Eksempel:

```
interface ISpiller {
  void utførTrekk(Spillebrett brett);
}
```

- Man ikke opprette objekter av `ISpiller`-kontrakten.
- Man kan lage objekter av klasser som implementerer `ISpiller`-kontrakten.
- Man kan referere til disse objektene med felt og lokale variabler av referansetypen `ISpiller`.
- Det kan finnes mer enn en implementasjon som oppfyller `ISpiller`-kontrakten.

## To implementasjoner av en kontrakt

```
public class TerminalSpiller implements ISpiller {
  SpillTekstGrensesnitt grensesnitt;

  TerminalSpiller(SpillTekstGrensesnitt grensesnitt) {
    this.grensesnitt = grensesnitt;)
  }

  public void utførTrekk(Spillebrett brett) {
    assert !brett.erSpillFerdig();
    assert brett == grensesnitt.brett;
    brett.slippBrikkeNedKolonne(brett.nesteBrikke(), grensesnitt.beOmKolonneValg());
  }
}
```

---

```
public class RandomSpiller implements ISpiller {
  public void utførTrekk(Spillebrett brett) {
    assert !brett.erSpillFerdig();
    int valg;
    do {
      valg = (int) (Math.random() * Spillebrett.KOLONNE_ANTALL);
    } while (!brett.erGyldigKolonneValg(valg));
    brett.slippBrikkeNedKolonne(brett.nesteBrikke(), valg);
  }
}
```



## Klassebibliotek

- Kildekode kan ofte utnyttes i flere programmer.
- Kode som man identifiserer som generelt nyttig blir vanligvis skrevet om til klassebiblioteker.

- Eksempler på klassebibliotek:
  - JUnit.
  - Java sitt standard klassebibliotek.

- Klassebiblioteker bør utformes med omhu.
- Biblioteker bør:
  - være lette å forstå og bruke
  - kunne brukes uten å endre på kode i biblioteket
  - ikke uten god grunn begrense situasjonene hvor det er mulig å bruke biblioteket

## Import av klassenavn

- Uten import setning må klasser oppgies med fullt navn:
```
java.util.Scanner inndata = new java.util.Scanner();
```
- Import av klassenavn gjør bruk av fullt navn unødvendig:
```
import java.util.Scanner
...
Scanner inndata = new Scanner();
```
- Man kan importere alle klassenavnene fra en pakke samtidig:
```
import java.util.*;
...
Scanner inndata = new Scanner();
Formatter formatter = new Formatter();
```



## Import av statiske medlemsnavn

- Uten importsetningen må statiske felt og metoder refereres til med navnet til klassen medlemmene befinner seg i:
```
long fib = Math.round(Math.pow(0.5 + Math.sqrt(5)/2, n)/Math.sqrt(5));
```
- Import av medlemsnavn gjør det unødvendig å oppgi klassenavnet:
```
import static java.lang.Math.sqrt;
...
long fib = Math.round(Math.pow(0.5 + sqrt(5)/2, n)/sqrt(5));
```
- Man kan importere alle statiske medlemsnavn fra en klasse samtidig:
```
import static java.lang.Math.*;
...
long fib = round(pow(0.5 + sqrt(5)/2, n)/sqrt(5));
```

## Utforming bibliotek og rammeverk

- Organisér klassene i pakker:
  - En pakkedeklarasjon kan angis på starten av kildekodefiler for å deklarere at alle klasser, kontrakter og oppramstyper deklarert i filen tilhører den angitte pakken.

```
package spill.terminal.LettSpill;  // Fil: spill/terminal/LettSpill.java
public class LettSpill { /* ... */ };
```

  - Kompilering:

```
> javac spill/terminal/LettSpill.java
```

  - Kjøring:

```
> java -ea spill.terminal.LettSpill
```

- Et *rammeverk* er et klassebibliotek som i tillegg til å ta imot innkommende metodekall også utfører utgående metodekall.
- Rammeverk tillater definering nye klasser utenfor klassebiblioteket, som har metoder som vil bli utført når rammeverket gjør visse operasjoner.



## Innkapsling og adgangsmodifikatorer for pakkemedlemmer

- *Innkapsling* (eng. *encapsulation*) er en abstraksjonsteknikk.
- Innkapsling hjelper oss med å redusere kompleksiteten til koden.
- Utform programkode hvor essensiell informasjon er lett tilgjengelig.
  - Kun relevant informasjon bør være synlig utad.
  - Skjul detaljer.
- Operasjonene og dataene de jobber på bør være sterkt sammenknyttet.
  - Fortell andre objekter hva de skal gjøre.
  - Ikke spør dem om tilstand.
- En enkel form for innkapsling er å skjule informasjon ved bruk av *adgangsmodifikatorer*.

| **Adgangs-kontekst** | `public` | **Pakkesynlighet: ingen adgangs-modifikator** |
|---|---|---|
| I samme pakke | Ja | Ja |
| I andre pakker | Ja | Nei |

## Adgangsmodifikatorer for klassemedlemmer

- Adgangsmodifikatorene `public`, `protected` og `private` har blitt brukt for å skjule felter og metoder i klassen som ikke behøver å være allment tilgjengelig.

| **Adgangs-kontekst** | `public` | `protected` | **pakkesynlighet: ingen adgangs-modifikator (package)** | `private` |
|---|---|---|---|---|
| I samme klasse | Ja | Ja | Ja | Ja |
| I andre klasser i samme pakke | Ja | Ja | Ja | Nei |
| Fra subklasser i andre pakker | Ja | Ja | Nei | Nei |
| Fra annen kode i andre pakker | Ja | Nei | Nei | Nei |



## Programmerbart grensesnitt

- Programmerbart grensesnitt (eng. *Application Programming Interface*, API)
  - Den eksternt synlige delen av et klassebibliotek
- Klassebibliotek brukes av mange bør være «stabilt».
  - Programkode som bruker APIet må sjeldent skrives om.
- Endringer man kan gjøre i et klassebibliotek, uten fare for å introdusere kompileringsfeil i programkode som bruker APIet til klassebiblioteket:
  - legge til helt nye pakker, kontrakter, klasser.
  - legge til nye felter, metoder og konstruktører til klasser.
  - Slette felter og metoder som ikke er synlige, dvs. alle medlemmer som ikke er `public` eller `protected`.
  - Slette eller endre klasser som ikke er synlige, dvs. alle klasser som ikke er `public`.
  - Gjøre felter og metoder mer synlige ved å endre adgangsmodifikatoren.
  - Endre implementasjonen av metoder.