EDAA20
Programmering och databaser

Föreläsning 7 – Klasser

2023-09-18, Niklas Fors

Från **Kursplanen** (se Moodle-sida)

Programmering

	Läsvecka 1	Läsvecka 2	Läsvecka 3	Läsvecka 4	Läsvecka 5	Läsvecka 6	Läsvecka 7	
Mån	F1	F3	F5	F7	F9	F11	F13	
Tis	F2	F4	F6	F8	F10	F12	F14	
Ons	L1	L3	L4	L6	L8	L	L11	
Fre	L2	L	L5	L7	L9	L10	L	Instuderings-vecka
Ons	R	R	R	R	R	R	R	

F föreläsningar, L datorlaborationer (obligatoriska), R resurstid

Nu är vi färdiga med databasdelen!

Mer arbete krävs nu mellan labbpassen

Tenta, måndag 23 oktober

Hur klarar man kursen?

- **Man behöver träna och programmera mycket!**
- Kontinuerligt arbete
  - 50% heltidsstudier, runt 20 timmar/veckan (inkl. föreläsningar, självstudier osv)
- Följ föreläsningar
  - Koncept introduceras
- Lös Moodle-uppgifter
  - Mindre uppgifter om enskilda koncept
- Lös labbuppgifter
  - Större uppgifter där koncept kombineras
  - Kluriga – fråga labbledare om hjälp när ni fastnar!
  - Fokusera på förståelse

Objekt:

- Består av attribut och metoder
- Skapa: `new Klassnamn(argument)`
- Metodanrop: `ref.metodnamn(argument)`
- Definieras av en klass

Klass:

- Mall för hur objekt skapas
- Definierar attribut (`private typ namn`)
- Definierar konstruktor (anger startvärden för attributen)
- Definierar metoder (har returtyp, namn och parametrar)

```java
Square sq = new Square(10, 20, 40);  // Skapa kvadratobjekt
sq.move(5, 5);                       // Metodanrop med två argument
int a = sq.getX();
```

```java
public class Square {                // Klassen Square
  private int x;                     // Attribut
  ...

  public Square(int x, ...) {        // Konstruktor med parameter
    this.x = x;                      // Initiera attribut
    ...
  }

  public void move(int dx, int dy) { // Metod med två parametrar
    x = x + dx;
    y = y + dy;
  }

  ...
}
```

Övning: Vad gör följande program?

```
public class A {
  public static void main(String[] args) {
    Random rand = new Random();
    int n = 1;
    int x = rand.nextInt(6) + 1;
    while (x != 6) {
      x = rand.nextInt(6) + 1;
      n++;
    }
    System.out.println(n);
  }
}
```

Övning: Vad gör följande program?

```
public class NbrRolls {
  public static void main(String[] args) {
    Random rand = new Random();
    int nbrRolls = 1;
    int x = rand.nextInt(6) + 1;
    while (x != 6) {
      x = rand.nextInt(6) + 1;
      nbrRolls++;
    }
    System.out.println("Det krävdes " + nbrRolls + " kast.");
  }
}
```

Programmet simulerar en tärning och räknar antalet kast tills man får en sexa.

Tärningsmodell

För att simulera ett tärningskast skriver vi:

```
x = rand.nextInt(6) + 1;
```

Det är lite svårläst vad detta betyder och det är lätt att glömma bort +1

Ett sätt att lösa detta är att introducera en klass som modellerar tärningar

Specifikation

Specifikation av klassen `Dice`:

```
/** Skapar en tärning med 6 sidor */
Dice();

/** Kastar tärningen */
void roll();

/** Returnerar det senaste kastet */
int getDots();
```

Specifikation

- En ***specifikation*** beskriver hur en klass används (hur man skapar objekt, vilka metoder som finns och hur de används). Detta beskrivs typiskt i dokumentation för klassen.
- En specifikation ses som ett kontrakt mellan klassens implementatör och de som använder klassen

Användning av klassen Dice

```
public class NbrRolls {
  public static void main(String[] args) {
    Dice d = new Dice();
    int nbrRolls = 1;
    d.roll();
    while (d.getDots() != 6) {
      d.roll();
      nbrRolls++;
    }
    System.out.println("Det krävdes " + nbrRolls + " kast.");
  }
}
```

Först skapas ett `Dice`-objekt

Därefter anropas metoder på `Dice`-objektet för att kasta tärningen och ta reda på senaste kastet

Givet specifikationen av klassen `Dice`:

```
/** Skapar en tärning med 6 sidor */
Dice();

/** Kastar tärningen */
void roll();

/** Returnerar det senaste kastet */
int getDots();
```

Vilka attribut behöver klassen? (Vad behöver tärningsobjekten spara?)

Vilka startvärden ska attributen få i konstruktorn?

Givet specifikationen av klassen `Dice`:

```
/** Skapar en tärning med 6 sidor */
Dice();

/** Kastar tärningen */
void roll();

/** Returnerar det senaste kastet */
int getDots();
```

Vilka attribut behöver klassen? (Vad behöver tärningsobjekten spara?)

- Senaste kastet (heltal)
- Slumptalsgenerator (`Random`)

Vilka startvärden ska attributen få i konstruktorn?

Implementation av Klassen Dice

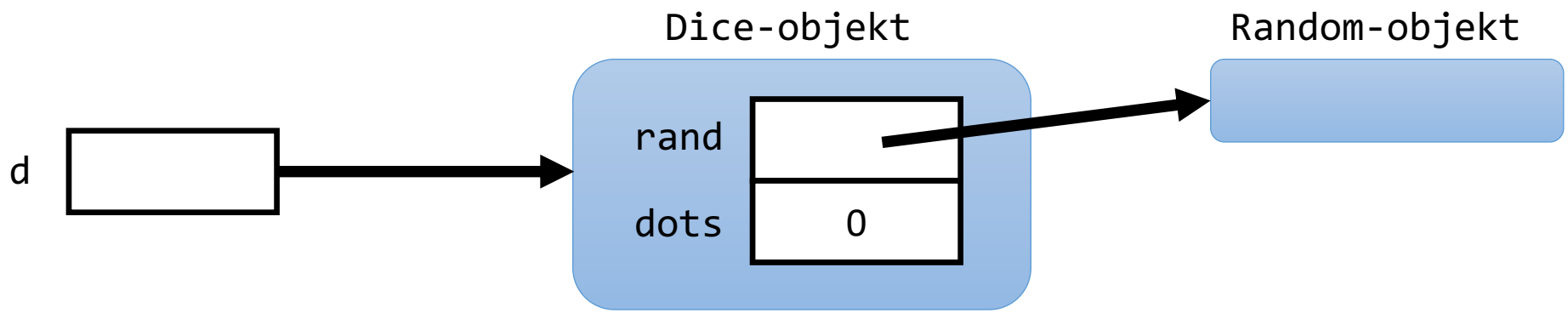
```
public class Dice {
  private Random rand; // Slumptalsgenerator
  private int dots;    // Senaste kastet

  /** Skapar en tärning med 6 sidor */
  public Dice() {
    rand = new Random();
    dots = 0;          // Tärning ej kastad än. Initiera till 0
  }

  /** Kastar tärningen. */
  public void roll() {
    dots = rand.nextInt(6) + 1;
  }

  /** Tar reda på vad tärningen visar. */
  public int getDots() {
    return dots;
  }
}
```

Tärningsobjekt i minnet

```
Dice d = new Dice();
d.roll();
System.out.println(d.getDots());
```

Tärningsobjekt i minnet – exekvering 1/3

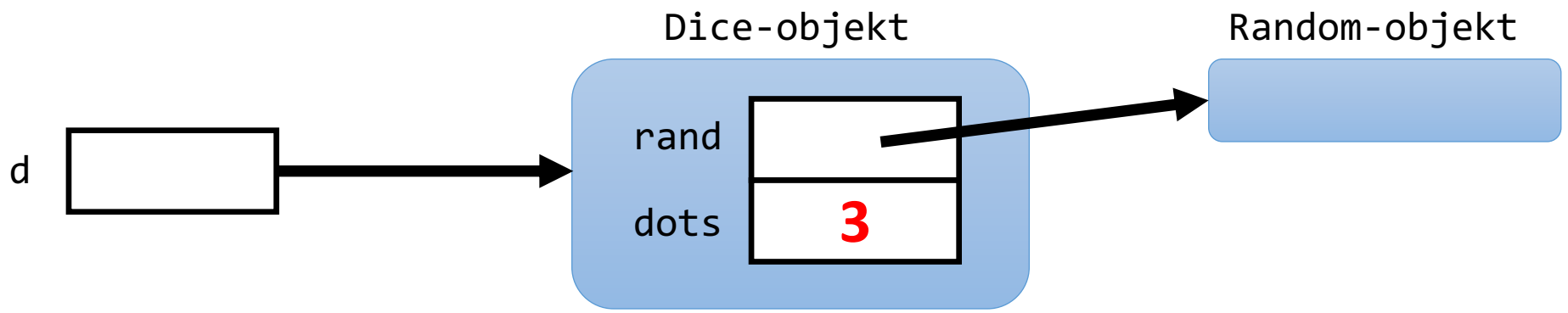
- `Dice d = new Dice();` Tärningsobjekt skapas och attributen initieras
  `d.roll();`
  `System.out.println(d.getDots());`



Tärningsobjekt i minnet – exekvering 2/3

```
Dice d = new Dice();
d.roll();
System.out.println(d.getDots());
```

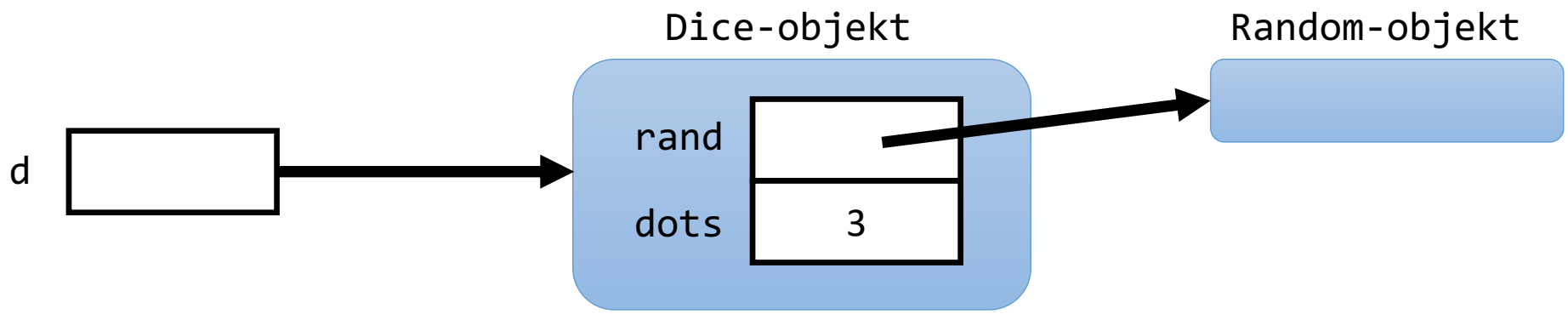
Tärningen kastas och blir **exempelvis 3**, vilket sparas i attributet dots



Tärningsobjekt i minnet – exekvering 3/3

```
Dice d = new Dice();
d.roll();
System.out.println(d.getDots());
```

Senaste kastet hämtas (3) och skrivs ut



Antalet kast?

```
public class NbrRolls {
  public static void main(String[] args) {
    Dice d = new Dice();
    int nbrRolls = 1;
    d.roll();
    while (d.getDots() != 6) {
      d.roll();
      nbrRolls++;
    }
    System.out.println("Det krävdes " + nbrRolls + " kast.");
  }
}
```

Vi kan låta tärningen själv hålla reda på antalet kast

Klassen Dice

```
public class Dice {
  private Random rand;
  private int dots;

  /** Skapar en tärning med 6 sidor */
  public Dice() {
    rand = new Random();
    dots = 0;
  }

  /** Kastar tärningen. */
  public void roll() {
    dots = rand.nextInt(6) + 1;
  }

  /** Tar reda på vad tärningen visar. */
  public int getDots() {
    return dots;
  }
}
```

ÖVNING

Vilka ändringar behövs för att hålla reda på antalet kast?

Behövs fler attribut? Startvärde?

Behövs metoder ändras?

Behövs fler metoder?

Klassen Dice

```
public class Dice {
  private Random rand;
  private int dots;

  /** Skapar en tärning med 6 sidor */
  public Dice() {
    rand = new Random();
    dots = 0;
  }

  /** Kastar tärningen. */
  public void roll() {
    dots = rand.nextInt(6) + 1;
  }

  /** Tar reda på vad tärningen visar. */
  public int getDots() {
    return dots;
  }
}
```

ÖVNING

Vilka ändringar behövs för att hålla reda på antalet kast?

Behövs fler attribut? Startvärde?

- Antalet kast (heltal) med 0 som startvärde

Behövs metoder ändras?

- Metoden `roll` behöver öka antalet kast med ett

Behövs fler metoder?

- För att returnera antalet kast

```java
public class Dice {
  private Random rand; // Slumptalsgenerator
  private int dots;    // Senaste kastet
  private int rolls;   // Antalet kast

  public Dice() {
    rand = new Random();
    dots = 0;
    rolls = 0;
  }

  public void roll() {
    dots = rand.nextInt(6) + 1;
    rolls++;
  }

  public int getNbrRolls() {
    return rolls;
  }

  ...
}
```

Ändringar i klassen `Dice` för att räkna antalet kast

Användning

```
public class NbrRolls {
  public static void main(String[] args) {
    Dice d = new Dice();
    d.roll();
    while (d.getDots() != 6) {
      d.roll();
    }
    System.out.println("Det krävdes " + d.getNbrRolls() + " kast.");
  }
}
```

Nu håller tärningen reda på antalet kast och
metoden `getNbrRolls()` anropas för att hämta antalet kast

Övning: Generalisering

Klassen `Dice` hanterar tärningar med sex sidor

Hur *generaliserar* vi klassen för att hantera godtyckligt många sidor?

- **Behövs fler attribut?**
- **Behövs konstruktorn ändras?**
- **Behövs metoderna ändras?**
- **Behövs fler metoder?**

Övning: Generalisering

Klassen `Dice` hanterar tärningar med sex sidor

Hur *generaliserar* vi klassen för att hantera godtyckligt många sidor?

- **Behövs fler attribut?**
  - Antalet sidor (heltal)
- **Behövs konstruktorn ändras?**
  - Konstruktorn behöver en parameter som tilldelas attributet
- **Behövs metoderna ändras?**
  - Metoden `roll()` behöver ta hänsyn till antalet sidor
- **Behövs fler metoder?**
  - Kanske. Man skulle kunna lägga till en metod som returnerar antalet sidor

```java
public class Dice {
  private Random rand;
  private int dots;
  private int rolls;
  private int sides;

  public Dice(int numSides) {
    rand = new Random();
    dots = 1;
    rolls = 0;
    sides = numSides;
  }

  public void roll() {
    dots = rand.nextInt(sides) + 1;
    rolls++;
  }

  ...
}
```

Ändringar i klassen `Dice` för att hantera godtyckligt antal sidor.

Man skulle kunna lägga till en metod `getSides` för att returnera antalet sidor.

```java
public class Dice {
  private Random rand;
  private int dots;
  private int rolls;
  private int sides;

  public Dice(int sides) {
    rand = new Random();
    dots = 1;
    rolls = 0;
    this.sides = sides;
  }

  public void roll() {
    dots = rand.nextInt(sides) + 1;
    rolls++;
  }

  ...
}
```

Det är vanligt att ha samma parameternamn som attributnamn.

I tilldelningen måste man då använda `this.sides` för att hänvisa till attributet `sides` och inte parametern `sides`.

Användning

```
public class NbrRolls {
  public static void main(String[] args) {
    Dice d1 = new Dice(12);
    Dice d2 = new Dice(6);
    d1.roll();
    d2.roll();
    System.out.println("Första tärningen: " + d1.getDots());
    System.out.println("Andra tärningen: " + d2.getDots());
  }
}
```

Olika tärningar med olika antal sidor

Flera konstruktorer

- När man skapar ett objekt av den generella tärningsklassen måste man nu ange antalet sidor:

```
new Dice(6)
```

- Normalfallet är dock att tärningar har 6 sidor. För att förenkla användningen av klassen kan vi lägga till en konstruktor som skapar en tärning med 6 sidor.

```
/** Skapar en tärning med 6 sidor */
public Dice() { ... }
/** Skapar en tärning med sides sidor */
public Dice(int sides) { ... }
```

```java
public class Dice {
  private Random rand;
  private int dots;
  private int rolls;
  private int sides;

  public Dice() {
    rand = new Random();
    dots = 1;
    rolls = 0;
    sides = 6;
  }

  public Dice(int sides) {
    rand = new Random();
    dots = 1;
    rolls = 0;
    this.sides = sides;
  }

  ...
}
```

En till konstruktor som skapar en tärning med sex sidor:
`new Dice()`

```java
public class Dice {
  private Random rand;
  private int dots;
  private int rolls;
  private int sides;

  public Dice() {
    this(6);



  }

  public Dice(int sides) {
    rand = new Random();
    dots = 1;
    rolls = 0;
    this.sides = sides;
  }

  ...
}
```

Istället för att repetera koden i den nya konstruktorn anropar vi den andra konstruktorn

Man kan i början av en konstruktor anropa en annan konstruktor: `this(argument)`

Överlagring – Flera konstruktorer

- En klass kan således ha flera konstruktorer

```
/** Skapar en tärning med 6 sidor */
public Dice() { ... }
/** Skapar en tärning med side sidor */
public Dice(int sides) { ... }
```

- Konstruktorerna måste ha olika parametrar (att antalet parametrar skiljer sig åt eller parametertyperna)
- Detta kallas ***överlagring*** (eng: overloading)
  - Gäller även för vanliga metoder, dvs, att två metoder kan ha samma namn fast olika parametrar

Ange startvärden åt attribut direkt

```
public class Dice {
  private Random rand = new Random();
  private int dots = 1;

  /** Kastar tärningen. */
  public void roll() {
    dots = rand.nextInt(6) + 1;
  }

  /** Tar reda på vad tärningen visar. */
  public int getDots() {
    return dots;
  }
}
```

Det är möjligt att ange startvärden för attribut direkt när man deklarerar attributen.

Anges ingen konstruktor får klassen automatiskt en tom *default*-konstruktor:

```
public Dice() { }
```

Scope

Scope syftar på var en variabel får användas (attribut, parameter, osv)

```
public class Classname {
  private int attr;

  public void m(int p) {
    int x = attr*2;
    for (int i = 0; i < p; i++) {
      if (i < x) {
        int y = i+2;
        ...
      }
    }
    ...
  }
  ...
}
```

Attributet **`attr`** får användas i hela klassen

Parametern **`p`** får användas i metoden **`m`**

Variabeln **`x`** får användas efter deklarationen och i metoden **`m`**

Variabeln **`i`** får användas i for-satsen

Variabeln **`y`** får användas i if-satsen

Typiskt får en variabel användas i det block ({}) den är deklarerad i.

Olika sorters fel

- **Kompileringsfel** – bryter mot språkets regler
  - Glömt ett {
  - Glömt att deklarera en variabel innan den används
  - …
- **Exekveringsfel** (runtime errors) – programmet kraschar under exekvering
  - `ArrayIndexOutOfBoundException`
  - `NullPointerException`
  - …
- **Logiska fel** – programmet körs men beräknar fel sak
  - Minus (-) istället för plus (+)
  - Oavsiktliga heltalsdivision
  - För tidig return - undersöker bara första elementet vid sökning i en vektor
  - …

Felsökning

Några tips:

- Programmera stegvis och testa ofta!
- Läs felmeddelandet! Vid exekveringsfel, börja läsa uppifrån.
- Lägg till utskriftsrader (`println`) för att förstå vad som händer
- Använd debugger

Kursboken *Think Java* beskriver detta på ett bra sätt. Se appendix C