EDAA20
Programmering och databaser

Föreläsning 6 – Klasser

2023-09-12, Niklas Fors

Objekt

- Objekt har ***attribut*** och ***metoder***
- Skapa objekt: **`new`** `Klassnamn(`*`argument`*`)`

- **Referensvariabler** lagrar referenser till objekt
  - Referensen kopieras vid tilldelning (=)
  - Referensen jämförs vid jämförelse (==)

Klass

En ***klass*** definierar hur objekt kan skapas och användas

En klass definierar:

- ***Attribut***
- ***Konstruktorer*** - särskilda metoder som ger startvärden till attributen
- ***Metoder***

En klass lagras normalt i en egen fil

Kvdratobjekt

Igår skapade vi Square-objekt:

```
Square sq = new Square(20, 10, 40);
int x = sq.getX();
int y = sq.getY();
System.out.println(x + ", " + y);
```

Idag ska vi implementera klassen Square

Kvdratobjekt

Igår skapade vi Square-objekt:

```
Square sq = new Square(20, 10, 40);
```

Vilka attribut (namn och typ) har ett Square-objekt?

Vilka värden ska attributen ha från start?

Kvdratobjekt

Igår skapade vi Square-objekt:

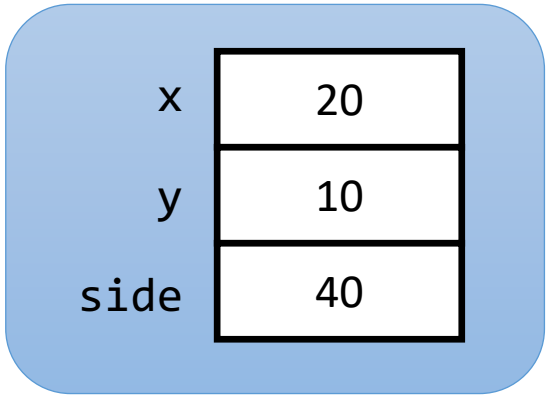
```
Square sq = new Square(20, 10, 40);
```

Vilka attribut (namn och typ) har ett Square-objekt?

- `int x`
- `int y`
- `int side`

Vilka värden ska attributen ha från start?

- Attributet x = första argumentet (20)
- Attributet y = andra argumentet (10)
- Attributet `side` = tredje argumentet (40)



Definieras i konstruktorn

Klass

En ***klass*** definierar vilka attribut, konstruktorer och metoder ett objekt har:

```
public class Klassnamn {
  // Attribut (deras typ och namn)

  // Konstruktor(er) – Anger startvärden till attributen.
  //                   Har samma namn som klassen.

  // Metoder
}
```

Attribut i klassen `Square`

```
public class Square {
  // Attribut
  private int x;
  private int y;
  private int side;

  ...
}
```

Attribut:

- Allokeras när objektet skapas och får först standardvärden (0, 0.0, false, null, …)
- Ges normalt startvärden i konstruktorn
- Får användas i hela klassen
- Deklareras normalt som `private` ⟹ kan enbart användas inuti klassen

Konstruktor – hur objekt initieras

```
public class Square {
  private int x;
  private int y;
  private int side;

  public Square(int xPos, int yPos, int sideLength) {
    x = xPos;
    y = yPos;
    side = sideLength;
  }
}
```

Konstruktor:

- Anger startvärden till attribut (ofta utifrån parametrar)
- Har samma namn som klassen och saknar returtyp
- Anropas automatiskt när ett objekt skapas

`public` eller `private`?

- Man behöver ange synlighet för attribut och metoder:
  - **`private` – kan bara användas inuti klassen**
  - **`public` – kan användas inuti och utanför klassen**

- Tumregel:
  - **Privata attribut**
  - **Publika metoder**

- Några undantag:
  - Metoder som bara används i klassen kan vara privata
  - Attribut som är konstanter kan vara publika (`Math.PI` osv)

Skapa objekt

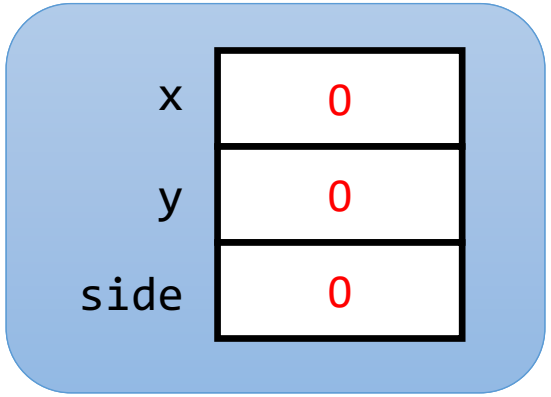
```
new Square(20, 30, 40)
```

```
public class Square {
  private int x;
  private int y;
  private int side;

  public Square(int xPos, int yPos, int sideLength) {
    x = xPos;
    y = yPos;
    side = sideLength;
  }
}
```

Skapa objekt – exekvering (1/3)

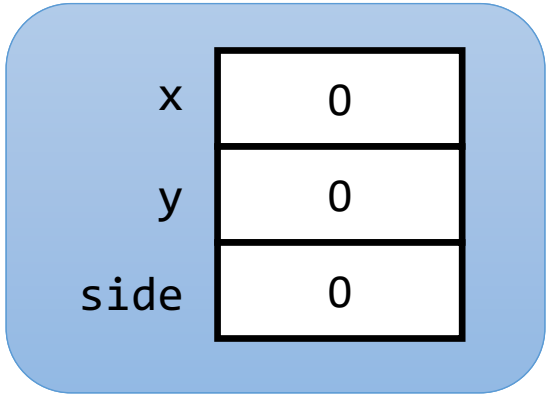
(1) Plats allokeras för objektet i minnet och alla attribut får standardvärden (0, 0.0, false, null, osv)

```
new Square(20, 30, 40)
```




```
public class Square {
  private int x;
  private int y;
  private int side;

  public Square(int xPos, int yPos, int sideLength) {
    x = xPos;
    y = yPos;
    side = sideLength;
  }
}
```

Skapa objekt – exekvering (2/3)



(1) Plats allokeras för objektet i minnet och alla attribut får standardvärden (0, 0.0, false, null, osv)

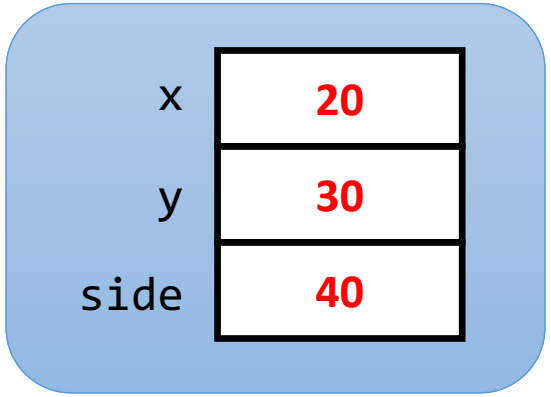
```
new Square(20, 30, 40)
```

```
public class Square {
  private int x;
  private int y;
  private int side;

  public Square(int xPos, int yPos, int sideLength) {
    x = xPos;
    y = yPos;
    side = sideLength;
  }
}
```

=20 =30 =40

(2) Argumenten överförs till parametrarna i konstruktorn

Skapa objekt – exekvering (3/3)



(1) Plats allokeras för objektet i minnet och alla attribut får standardvärden (0, 0.0, false, null, osv)

```
new Square(20, 30, 40)
```

(2) Argumenten överförs till parametrarna i konstruktorn

```
public class Square {
  private int x;
  private int y;
  private int side;

  public Square(int xPos, int yPos, int sideLength) {
    x = xPos;
    y = yPos;
    side = sideLength;
  }
}
```

=20 =30 =40

(3) Attributen tilldelas parametrarna

Några metoder

```
public class Square {
  private int x;
  private int y;
  private int side;

  ...

  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }

  public int getSide() {
    return side;
  }
}
```

Varje metod har:

- Synlighet (`public`)
- Returtyp
  - `void` om returvärde ej behövs
- Namn
- Noll eller flera parametrar
- Kodblock
  - return-sats krävs när returtypen inte är void

Attributen är åtkomliga i metoderna

Några fler metoder

```
public class Square {
  private int x;
  private int y;
  private int side;

  ...

  /** Returnerar arean */
  public int getArea() {
    return side*side;
  }

  /** Uppdaterar sidan till newSide */
  public void setSide(int newSide) {
    side = newSide;
  }
}
```

Parametern `newSide` används för att uppdatera attributet `side`

Övning – implementera metod

```
public class Square {
  private int x;
  private int y;
  private int side;

  ...

  /** Flyttar kvadraten relativt */
  public void move(int dx, int dy) {
    // ÖVNING: Implementera metoden

  }
}
```

Övning – lösning

```
public class Square {
  private int x;
  private int y;
  private int side;

  ...

  /** Flyttar kvadraten relativt */
  public void move(int dx, int dy) {
    x = x + dx;
    y = y + dy;
  }
}
```

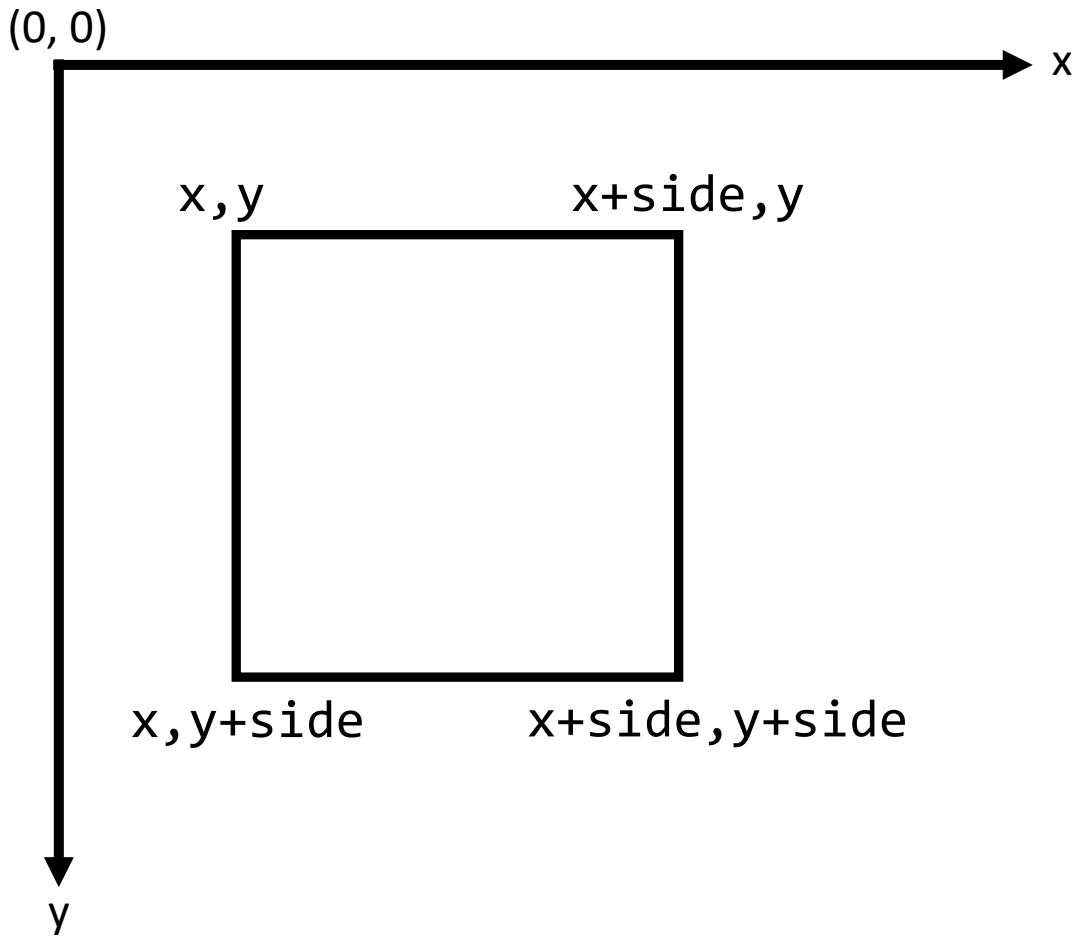
Rita kvadraten i ett fönster

```
public class Square {
  private int x;
  private int y;
  private int side;

  ...

  /** Ritar kvadraten i fönstret w */
  public void draw(SimpleWindow w) {




  }
}
```

Metoden `draw` tar ett fönsterobjekt som parameter och använder det för att rita kvadraten.

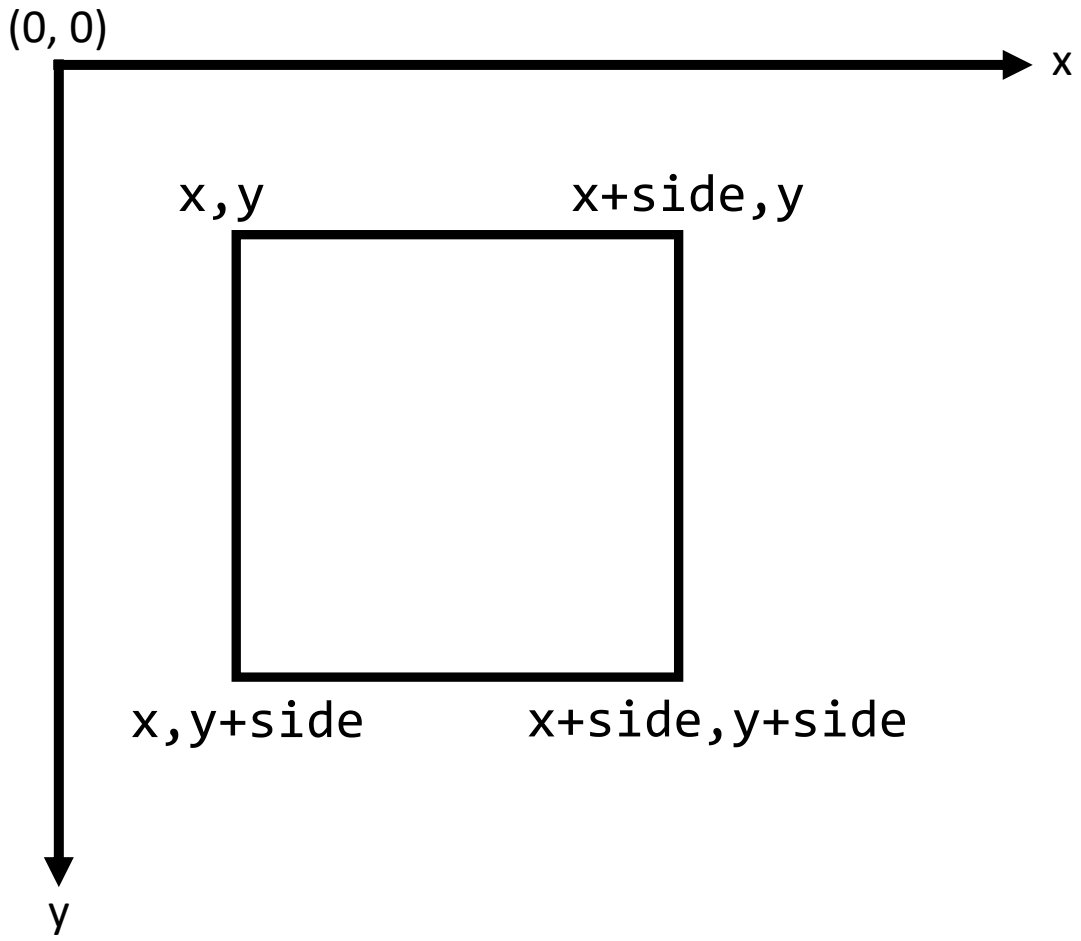
Rita kvadraten i ett fönster

```
public class Square {
  private int x;
  private int y;
  private int side;

  ...

  /** Ritar kvadraten i fönstret w */
  public void draw(SimpleWindow w) {




  }
}
```



Rita kvadraten i ett fönster

```
public class Square {
  private int x;
  private int y;
  private int side;

  ...

  /** Ritar kvadraten i fönstret w */
  public void draw(SimpleWindow w) {
    w.moveTo(x, y);
    w.lineTo(x, y + side);
    w.lineTo(x + side, y + side);
    w.lineTo(x + side, y);
    w.lineTo(x, y);
  }
}
```



Övning – implementera metod

```
public class Square {
  private int x;
  private int y;
  private int side;

  ...

  /** Returnerar true om kvadraten har större area än
    * kvadraten other, i annat fall returneras false. */
  public boolean biggerThan(Square other) {
    // ÖVNING: Implementera metoden




  }
}
```

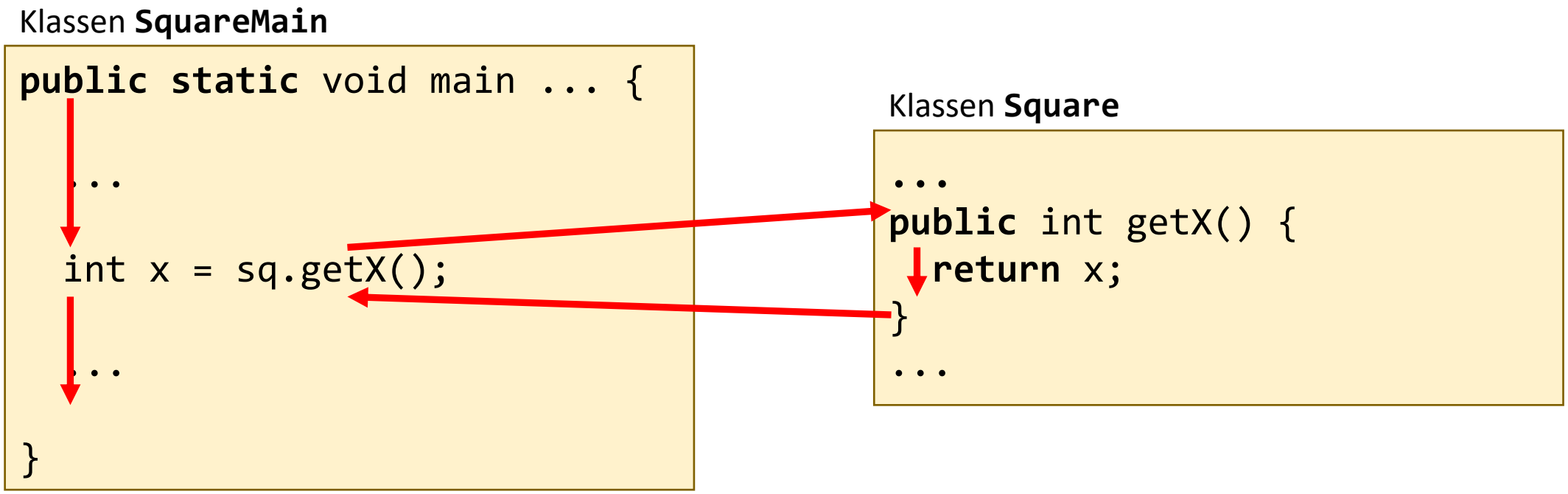
Övning – lösning

```
public class Square {
  private int x;
  private int y;
  private int side;

  ...

  /** Returnerar true om kvadraten har större area än
    * kvadraten other, i annat fall returneras false. */
  public boolean biggerThan(Square other) {
    if (getArea() > other.getArea()) {
      return true;
    } else {
      return false;
    }
  }
}
```

Övning – förenkling

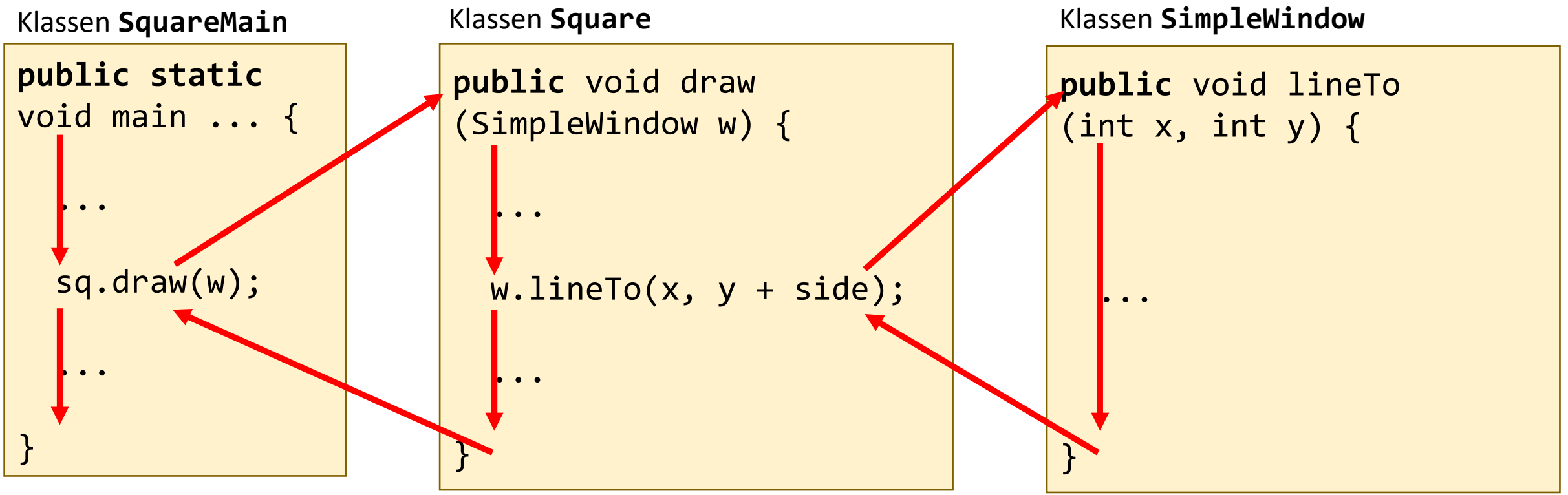
```
public class Square {
  private int x;
  private int y;
  private int side;

  ...

  /** Returnerar true om kvadraten har större area än
    * kvadraten other, i annat fall returneras false. */
  public boolean biggerThan(Square other) {
    return getArea() > other.getArea();
  }




}
```

Exekvering



- Ett metodanrop innebär att exekveringen fortsätter med kodblocket i den anropade metoden (`getX`)
- När den anropade metoden är klar återupptas exekveringen i den metod där anropet gjordes (`main`)

Exekvering – flera metodanrop



Metod:

```
public void move(int dx, int dy) {
   ...
}
```

Parametrar

Metodanrop:

```
sq.move(10, 20);
```

Argument

Parametrar:

- Är indata till metoden
- ***Argumenten*** i anropet **överförs** till ***parametrarna***
- Skapas när metoden anropas och tas bort när metoden är färdigexekverad
- Är enbart åtkomliga inuti metoden

Konstruktor – alternativ implementation

```
public class Square {
  private int x;
  private int y;
  private int side;

  public Square(int x, int y, int side) {
    this.x = x;
    this.y = y;
    this.side = side;
  }
}
```

`this` är en referens till objektet själv

- Ofta heter parametrarna i konstruktorn samma som attributen
- Då behöver man ange `this.` framför attributen för att särskilja dem från parametrarna

Statiska metod

För en statisk (`static`) metod behövs inget objekt

Exempelvis är alla metoder i klassen `Math` statiska:

```
double x = Math.sqrt(42);     // Kvadratroten
```

Checklista

- Förklara begreppen objekt och klass
- Skapa objekt och anropa metoder på dem
- Implementera klasser (med attribut, konstruktorer och metoder)
- Förstå vilka variabler och objekt som finns i ett program och hur de hänger ihop. Visa detta genom att rita en figur över minnessituationen med variabler och objekt med attribut.