```
public class Flerfamiljshus extends Bostadshus {
  int antalLägenheter;
  public static final double hyraPerM2 = 2000;

  public double beräknadHyresinkomst() {
    return yta() * hyraPerM2;
  }

  @Override
  public double yta() {
    return längd * bredd * antalVåningar * 0.95;
  }
}
```

Alternativ version:

```
  @Override
  public double yta() {
    return super.yta() * 0.95; // anropa yta() i klassen Hus
  }
```

```
Hus h = new Hus();
Flerfamiljshus f = new Flerfamiljshus();
h.bredd = 10; h.längd=20; h.antalVåningar = 3;
f.bredd = 10; f.längd=20; f.antalVåningar = 3;

System.out.println(h.yta());
System.out.println(f.yta());
```

Självklart:

```
600.0
570.0
```

Men

```
Hus h2;
h2 = f;

System.out.println(h2.yta());  // Vilken metod anropas?
```

Dynamisk bindning ger:

```
570.0
```

```
public class Hus {
  private double längd;
  private double bredd;
  private int antalVåningar;

  // Konstruktorer
  public Hus() {}

  public Hus(double l, double b, int v) {
    sättLängd(l); sättBredd(b); sättAntalVåningar(v);
  }

  // Instansmetoder
  public void sättLängd(double l) {
    if (l > 0)
      längd = l;
    else
      throw new IllegalArgumentException("Negativ längd");
  }

  etc.
}
```

Initiering av objekt:

1. Alla instansvariabler som deklareras i klassen sätts till sina defaultvärden (0 eller motsvarande).
2. En konstruktor för superklassen anropas.
3. De instansvariabler i klassen som har explicita initieringsuttryck initieras till dessa värden.
4. Satserna i subklassens konstruktor exekveras.

```
public class Bostadshus extends Hus {
  boolean tilläggsisolerat;

  // Konstruktorer
  Bostadshus(boolean isol) {
    // super() anropas automatiskt
    tilläggsisolerat = isol;
  }

  Bostadshus() {
    // super() anropas automatiskt
    tilläggsisolerat = true;
  }

  Bostadshus(double l, double b, int v, boolean isol) {
    super(l, b, v);         // måste ligga först
    tilläggsisolerat = isol;
  }

  // Instansmetoder
  public void isolera() {
    tilläggsisolerat = true;
  }
}
```

```
Hus[] ha = new Hus[100];
ha[0] = new Hus(40, 25, 4);
ha[1] = new Bostadshus(40, 25, 4, true);
ha[2] = new Flerfamiljshus(40, 25, 4, true, 10);
etc.


for (int i=0; i<ha.length; i++)
  if (ha[i] != null)
    System.out.println("Ett " + ha[i].getClass().getName()
                       + " med ytan " + ha[i].yta());
```

Eller

```
for (Hus h : ha)
  if (h != null)
    System.out.println("Ett " + h.getClass().getName()
                       + " med ytan " + h.yta());
```

```
Ett Hus med ytan 4000.0
Ett Bostadshus med ytan 4000.0
Ett Flerfamiljshus med ytan 3800.0
```

```
List<Hus> hl = new ArrayList<>();

hl.add(new Hus(40, 25, 4));
hl.add(new Bostadshus(40, 25, 4, true));
hl.add(new Flerfamiljshus(40, 25, 4, true, 10));


for (Hus h : hl)
  if (h != null)
    System.out.println("Ett " + h.getClass().getName()
                       + " med ytan " + h.yta());
```

```
public abstract class Hus {
  som tidigare
}
Hus h1 = new Hus();   // FEL!! Hus är en abstrakt klass
Hus h2 = new Flerfamiljshus();   // OK
```

```
public abstract class Djur {
  public abstract void rita();  // abstrakt metod
  övriga variabler om metoder
}


public class Tiger extends Djur {
  @Override
  public void rita() {
    här ligger satser som ritar en tiger
  }
  övriga variabler om metoder
}



Djur d = ...; // refererar till något djur, t.ex. en Tiger
d.rita();     // kan alltid utföras
```