

Åtkomsträttigheter och inkapsling

Två betydelser hos...

Inkapsling (Encapsulation)

Koppla ihop data och funktioner till objekt

- Fundamentalt i OO
- “Objekt: en **kapsel** som innehåller både data och funktioner”

Begränsa tillgång till medlemmar (fält, metoder, ...)

- Genom *accessmodifierare*
- “Vissa medlemmar är instängda i en **ogenomskinlig kapsel**, och kan inte ses eller utnyttjas från utsidan”



Att gömma data i Java



- I Java kan medlemmar ha dessa accessnivåer:
 - **public** – kod i **alla** klasser har tillgång
 - **protected** – tillgång i **underklasser** + klasser i samma **paket**
 - **[inget]** – tillgång i samma **paket** (bör undvikas)
 - **private** – tillgång bara inom samma **klass**

Vad hade
private betytt
utan OO?

```
public class Account
{
    private int accountNumber;
    private String password;
    ....
    public void deposit(...);
}
```

```
public class AccessTester {
    public void test(Account acc) {
        acc.number = 42; // forbidden!
        String pass = acc.password; // forbidden!
    }
}
```

Först: En exempelklass

Exempel 1: SortedList

- Vi vill skriva klassen **SortedList**
 - Lista som garanterar att elementen är i **sorterad ordning!**

```
class TestSortedList
{
    public static void main(String[] args) {
        SortedList letters = new SortedList();
        letters.add("Z");
        letters.add("A");
        System.out.println(letters);
        // [ "A", "Z" ]

        letters.add("M");
        System.out.println(letters);
        // [ "A", "M", "Z" ]
    }
}
```

Exempel 2: Implementation

- SortedList: En partiell implementation

```
class SortedList {  
    String[] elements;  
    int size;
```

Lagring för alla element: Array $\approx$ lista
(mer om detta i annan föreläsning)

```
String get(int pos) { return ... }
```

Hämtar element

```
void add(String el) {  
    int pos = findNewPositionFor(el);  
    insertAt(pos, el);  
}
```

Stoppar in element
på rätt plats
i arrayen...

```
int findNewPositionFor(String el) { ... }  
void insertAt(int pos, String el) { ... }  
}
```

... med dessa
hjälpmetoder
(nya begrepp!)

Varför inkapsling i detta exempel?

Inkapsling: För att skydda datastrukturer



```
/**  
 * This class maintains a sorted list of elements.  
 * Elements will always be in sorted order,  
 * regardless of the order in which they are added.  
 */
```

```
public class SortedList {  
    private String[] elements;  
    private int size;  
  
    String get(int pos) { return ... }  
    void add(String el) {  
        int pos = findNewPositionFor(el);  
        insertAt(pos, el);  
    }  
    int findNewPositionFor(String el) { ... }  
    void insertAt(int pos, String el) { ... }  
}
```

Detta är ett **kontrakt**
som vi måste uppfylla

Hur kan vi **garantera**
detta om **andra** kan
peta i våra interna
datastrukturer?

Privat → bara vår kod
kan modifiera data
→ vi kan se till att
kontraktet uppfylls

Inkapsling: För att uppfylla vårt kontrakt



```
/**  
 * This class maintains a sorted list of elements.  
 * Elements will always be in sorted order,  
 * regardless of the order in which they are added.  
 */
```

```
public class SortedList {  
    private String[] elements;  
    private int     size;  
  
    String         get(int pos)      { return ... }  
    void           add(String el)    {  
        int pos = findNewPositionFor(el);  
        insertAt(pos, el);  
    }  
  
    int            findNewPositionFor(String el) { ... }  
    private void   insertAt(int pos, String el) { ... }  
}
```

Även **insertAt()** kan bryta mot kontraktet om det inte anropas på korrekt sätt

Inkapsling: För överskådlighet

```
/**  
 * This class maintains a sorted list of elements.  
 * Elements will always be in sorted order,  
 * regardless of the order in which they are added.  
 */
```

```
public class SortedList {  
    private String[] elements;  
    private int size;  
  
    public String get(int pos) { return ... }  
    public void add(String el) {  
        int pos = findNewPositionFor(el);  
        insertAt(pos, el);  
    }  
  
    private int findNewPositionFor(String el) { ... }  
    private void insertAt(int pos, String el) { ... }  
}
```

Användaren behöver bara detta!

Om det andra syns blir klassen oöverskådlig!

Inkapsling: Minska åtaganden (kontrakt)



```
/**
 * This class maintains a sorted list of elements.
 * Elements will always be in sorted order,
 * regardless of the order in which they are added.
 */
public class SortedList {
    private String[] elements;
    private int size;

    public String get(int pos) { return ... }
    public void add(String el) {
        int pos = findNewPositionFor(el);
        insertAt(pos, el);
    }

    private int findNewPositionFor(String el) { ... }
    private void insertAt(int pos, String el) { ... }
}
```

Om hjälpmetoderna är publika, är de en del av vårt kontrakt



Andra kan skriva kod som anropar dem



Måste finnas kvar i all framtid!

Privat → bara vår kod kan anropa dessa
→ fritt fram att ändra!

Inkapsling: Minska åtaganden (kontrakt)

- Det som ingen annan ser kan vi ändra på

```
public class SortedList {  
    private String[] elements;  
    private int size;
```

```
    public String get(int pos) { return ... }  
    public void add(String el) {
```

```
        int pos = findNe  
        insertAt(pos, el);  
    }  
    private int f  
    private void i  
}
```

```
public class SortedList {  
    private ListNode first;  
    private int size;
```

```
    public String get(int pos) { return ... }  
    public void add(String el) {
```

```
        ListNode pos = findNewPositionFor(el);  
        insertElementAt(pos, el);  
    }
```

```
    private ListNode findNewPositionFor(String el) { ... }
```

```
    private void insertElementAt(ListNode pos, String el) { ... }  
}
```

Den "publika" delen har kvar
samma signatur
(metodnamn, parametrar, ...)



- Sammanfattning:
 - Ett stabilt publikt gränssnitt för andra att använda
 - Ett minimalt publikt gränssnitt
 - Minimera överflödigt funktionalitet som förvirrar
 - Minimera egna åtaganden och inlåsningsar inför framtiden
 - Ett skydd som tillåter klassen att garantera att kontraktet uppfylls

```
public class SortedList {  
    private String[] elements;  
    private int size;  
  
    public String get(int pos) { return ... }  
    public void add(String el) {  
        int pos = findNewPositionFor(el);  
        insertAt(pos, el);  
    }  
    private int findNewPositionFor(String el) { ... }  
    private void insertAt(int pos, String el) { ... }  
}
```

Inkapsling av fält

- Vanlig princip: **Fält** är implementationsdetaljer, **ska vara privata**
 - **Om** det verkligen är rimligt att andra klasser ska komma åt fältvärden:
 - Skapa en public **accessor**-metod (**getter**) vid behov
 - Skapa en public **mutator**-metod (**setter**) vid behov

Kan ändra intern representation:
Ändra bara getter/setter, som är i samma klass

- **private** double **diameter**;
/** Set the radius to r (must not be negative). */
public void setRadius(**final** double radius) {
 if (r >= 0.0) diameter = 2 * radius;
 else throw new **IllegalArgumentException**("Negative radius: " + r);
}

Kan lägga till konsistenskontroll

Namngivning: Getter, Setter

■ Namngivning:

```
public class Circle {  
    private double radius;  
    private boolean visible;  
  
    /** Set the radius to r (must not be negative). */  
    public void setRadius(final double r) {  
        if (radius >= 0.0) radius = r;  
        else throw new ...;  
    }  
  
    public double getRadius() {  
        return radius;  
    }  
    public boolean isVisible() {  
        return visible;  
    }  
}
```

Setters:
void setProperty(...)

Getters:
getProperty()

Booleska getters:
isProperty()

- Vissa anser att det finns undantag
 - ”Rena” datastrukturer
(inte mycket beteende, osannolikt att det skulle ändras i framtiden)

```
public class Dimension {  
    public int width;  
    public int height;  
}
```