

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:
 - **private** – tillgång bara inom samma **klass**

```
public class Account {
    private int accountNumber;
    private String password;
    ....
    public Money withdraw(int amount, String password) {
        // Can read this.password – within the same class
        if (password.equals(this.password)) {
            // Return some money
        } else {
            // Fail
        }
    }
}
```

Vad hade "private"
betytt utan OO?

```
public class AccessTester {
    public void tryToAccess(Account acc) {
        // Can't access private members of Account
        acc.number = 42; // forbidden!
        String pass = acc.password; // forbidden!
    }
}
```

Att gömma data i Java (2)

- I Java kan medlemmar ha dessa accessnivåer:
 - **private** – tillgång bara inom samma **klass**
 - `[inget]` – tillgång i samma **paket**
(bör undvikas, mest ett 'hack')
 - **protected** – tillgång i **underklasser** + klasser i samma **paket**
(ibland användbart vid ärvning – kommer senare)
 - **public** – kod i **alla** klasser har tillgång

```
public class Account {  
    public int accountNumber;  
    public String password;  
    ...  
}
```

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

Varför?

Hur ofta har vi så känslig information?

Först: En exempelklass

Exempel 1: SortedList



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

```
class SortedListTester
{
    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;  
  
    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) { ... }  
}
```

Lagring för alla element: Array

Hämtar element

Stoppar in element
på rätt sorterad
plats i arrayen...

... 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 *klassens kontrakt* om den 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) { ... }  
}
```

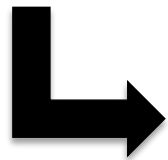
Användaren behöver bara detta!

Om det andra syns blir klassens API överskådligt!

Inkapsling: Minska åtaganden (kontrakt)

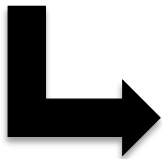
```
public class SortedList {  
    public int    findNewPositionFor(String el) { ... }  
    public void   insertAt(int pos, String el) { ... }  
}
```

Om vi tar vår kod...



FooBarDataStructures
library

och vi gör den allmänt
tillgänglig...



och många andra
använder den...

```
public class ProgramBySomeoneElse {  
    public void doSomethingUnexpected() {  
        int pos = sortList.findNewPositionFor("Hello");  
        ...  
    }  
}
```

kan de använda *allt* som
är publikt...

så om vi ändrar
beteendet *slutar deras
kod fungera!*

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 **privata**, är de **inte** en del av vårt kontrakt



Fritt fram för oss att ändra våra interna implementationsdetaljer!



Inkapsling: Minska åtaganden (kontrakt)

- Allt 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 = findNewPositionFor(el);  
        insertAt(pos, el);  
    }  
    private int findNewPositionFor(String el) { ... }  
    private void insertElementAt(int pos, String el) { ... }  
}
```

```
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 överflödig 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
 - Exempel:

```
public class Circle {  
    private double radius;  
  
    /** Set the radius to r (must not be negative). */  
    public void setRadius(final double radius) {  
        this.radius = radius;  
    }  
}
```

- Om vi vill göra en konsistenskontroll:

```
public class Circle {  
    private double radius;  
  
    /** Set the radius to r (must not be negative). */  
    public void setRadius(final double radius) {  
        if (radius < 0.0) {  
            throw new IllegalArgumentException("Negative radius: " + radius);  
        }  
        this.radius = radius;  
    }  
}
```

**Klassen har kontroll över sina objekt:
Kan bara ändra radie genom den egna
koden i setRadius()**

- Om vi vill ändra datarepresentation:

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

**Vi kan ha kvar setRadius()
i det publika gränssnittet
trots att vi ändrade intern lagring**

**”Användarnas” kod fungerar
precis som förut**

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(), has...()

- Vissa anser att det finns undantag
 - ”Ren datalagring”, klasser som används som poster (record/struct) (inte mycket beteende, osannolikt att det skulle ändras i framtiden)

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