

Föreläsning 21

Tobias Wrigstad

*Identitet \neq ekvivalens,
inkapsling, nästling, undantag*



Inkapsling

- Namnbaserad inkapsling

public, private och *package*

- Inkapsling och arv

protected

- Mer än bara namnbaserad inkapsling

Exempel med länkad lista och iterator

Identitet och ekvivalens

- Referenssemantik, värdesemantik
- Identitetsjämförelsen
- Ekvivalensjämförelsen

Signaturen för `equals()`

Overriding vs. overloading

`a.equals(b) == b.equals(a)?`

- `CompareTo<T>`

Wrapperklasser

- Skapa en kö med hjälp av arv

```
class Queue<T> extends List<T> {  
    /// ärver också insert(index, element)...  
}
```

Wrapperklasser

- Aggregering istället

```
class Queue<T> {  
    List<T> list = new LinkedList<T>();  
    void enqueue(T e) { list.add(e); }  
    T dequeue() {  
        T e = list.get(0);  
        list.remove(0);  
        return e;  
    }  
    ...  
}
```

Wrapperklasser

- Inkapsling?

```
class Queue<T> {  
    private List<T> list = new LinkedList<T>();  
    void enqueue(T e) { list.add(e); }  
    T dequeue() {  
        T e = list.get(0);  
        list.remove(0);  
        return e;  
    }  
    ...  
}
```

Inre och nästlade klasser

- Inre klass: **klass** nästlad inuti **instans**

```
class List { ... class Link { ... } ... }
```

”Varje lista har sin egen länkfabrik”: `List list = new List(); new list.Link();`

- Nästlad klass: klass nästlad inuti annan klass

```
class List { ... static class Link { ... } ... }
```

”Listorna delar samma länkfabrik via klassen”: `new List.Link();`

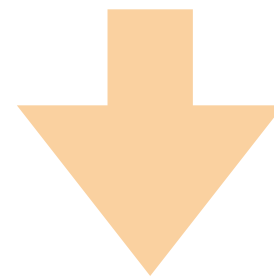
Undantagshantering

- I Java hanteras fel via exceptions
- Man kan själv ”kasta (throw) ett undantag (exception)”

```
throw new Exception()
```

- Flyttar kontrollflödet till **närmaste matchande omslutande catch-block**
- Exempel på hur exceptions kan fångas i c:

```
try {  
    Rectangle r = (Rectangle) someObject;  
    int x = y / z;  
} catch(ClassCastException e) {  
    ...  
} catch(ArithmeticException e) {  
    ...  
}
```



matchningsordning

Undantagsinformation propageras genom effekter

```
void postMessage(User u, Server s, Message m) {  
    Session session = s.logIn(u.id(), u.password());  
    session.post(m);  
}
```

```
void post(Message m) throws MalformedMessageException
```

- Om `MalformedMessageException` ärver av `Exception` är den ”**checked**”

Kräver att `postMessage` också **throws** `MalformedMessageException`, alternativt har ett catch-block runt anropet

- Om `MalformedMessageException` ärver av `RuntimeException` är den ”**unchecked**”

Behöver varken fångas eller explicit propageras

`jmf.NullPointerException`

Om MalformedMsg är ett Checked Exception

```
void post(Message m) throws MalformedMsgException
```

propagera

```
void postMessage(User u, Server s, Message m) throws MalformedMsgException {  
    Session session = s.logIn(u.id(), u.password());  
    session.post(m);  
}
```

eller

```
void postMessage(User u, Server s, Message m) {  
    Session session = s.logIn(u.id(), u.password());  
    try {  
        session.post(m);  
    } catch (MalformedMsgException e) {  
        ...  
    }  
}
```

hantera

Definiera egna exceptions

/// Checked

```
class MalformedMessageException extends Exception { ... }
```

/// Unchecked

```
class MalformedMessageException extends RuntimeException { ... }
```

Finally

```
void postMessage(User u, Server s, Message m) {  
    try {  
        Session session = s.logIn(u.id(), u.password());  
        session.post(m);  
    } catch (MalformedMessageException e) {  
        u.notify(...);  
    } finally {  
        session.logout();  
    }  
}
```

- Körs alltid, oavsett utgång i **try**-blocket
- Tillåter oss att lämna tillbaka resurser ("städa") oavsett vad som händer