

Testning och enhetstestning

En mycket kort introduktion till ett väldigt omfattande område...

Testning

- Varför testning? När?
- Stort område – yrkestitel: Testare
- Vem skriver testerna?
- Ett sätt att dela in tester
 - Enhetstester (unit tests)
 - Systemtester (integrationstester)
- En annan indelning
 - Black box
 - White box

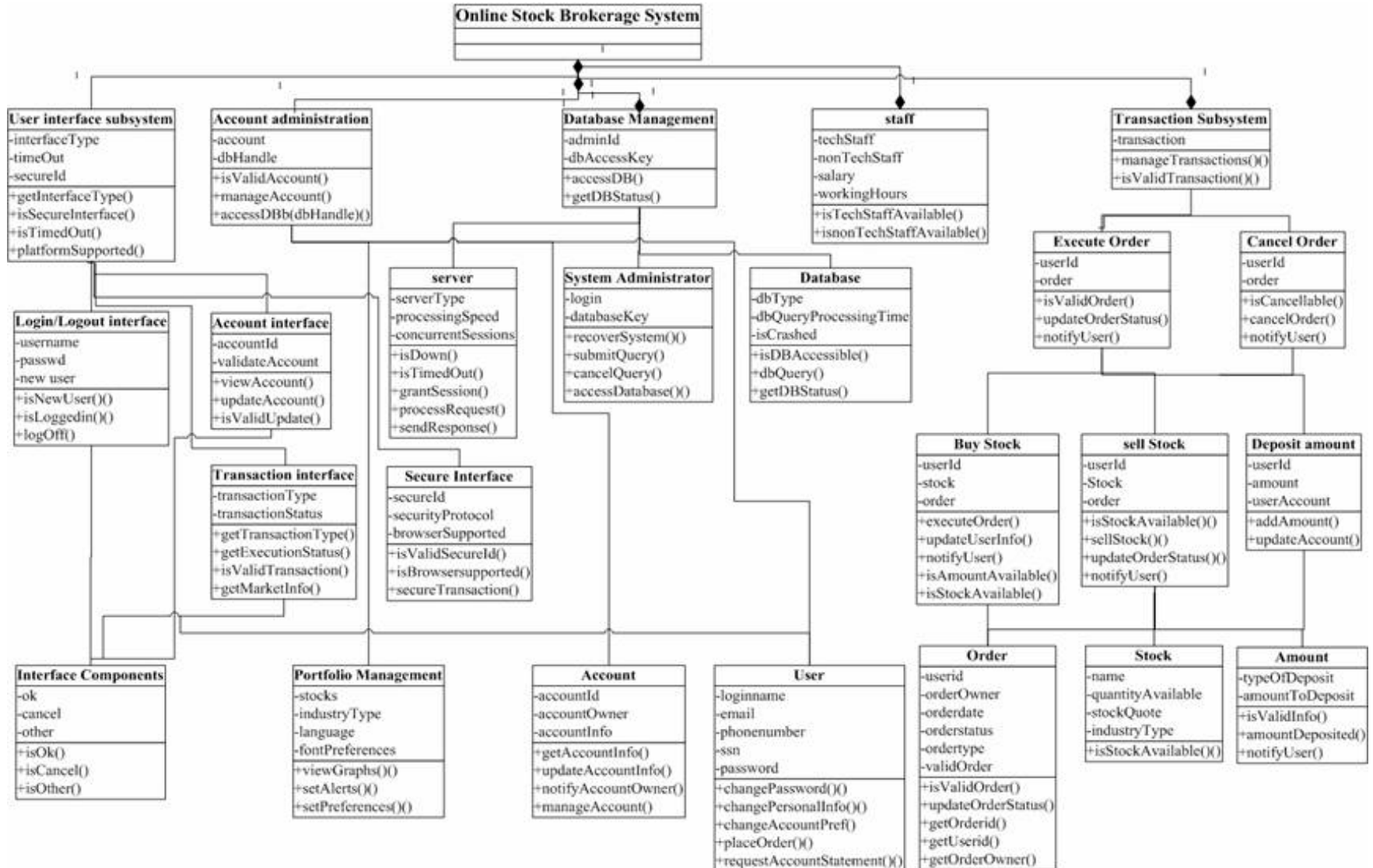
White vs black box-tester

- Black box
 - Designas utifrån kodens gränssnitt, utan vetskap om vald implementation
 - Möjligt att skriva tester före implementationen
- White box
 - Designas utifrån vetskap om implementationen
 - Testens struktur matchar kodens struktur – testa varje aspekt av koden, "code coverage"
- I båda fallen ovan måste vi vara klara över vad det förväntade resultatet är! (inte alltid trivialt)

Testning

- Enhet (i OOP)?
- Enhetstester
 - Fokuserar på klasser och individuella metoder
 - Säkerställer att enskilda kodeenheter uppför sig som de ska
 - Relativt lätta att skriva, exekvera och utvärdera
 - Testar (normalt) inte interaktionen med andra enheter
- System-/integrationstester
 - Testar att enheter fungerar tillsammans, interaktioner
 - Generellt svårare att skriva, exekvera och utvärdera

Enhetstester först, sedan integration...



Tester, att tänka på...

- Testen ska vara förutsägbara och upprepbara
 - Inget I/O i testkoden...
... om det inte är specifikt I/O som ska testas
 - Ersätt slumpade värden med valda värden (randvillkor et c.)
 - Ersätt beroenden till andra klasser/infrastruktur med "mock objects" - vi kan testa utan hela systemet samt styra över testfallen
- Glöm inte att testa även felaktigt indata
Vilka fall bör testas nedan?
 - ```
public String getItem(int index) {
 if (index >= n) throw new
 IndexOutOfBoundsException("index=" + index);

 return theItems[index];
}
```

# Tester, att tänka på...

- Code coverage – testa all funktionalitet
  - Alla metoder et c.
  - Alla "exekveringsvägar"

```
if(condition) {
 ...
} else {
 ...
}
```
- Randvillkor
  - Indata får ha värden [0..42] – vilka värden bör testas?
  - ```
private void pack(int index)  
    for(int i = index; i < n; i++) // n <= arr.length  
        arr[i] = arr[i+1];  
    n -= 1;  
}
```

I praktiken...

- Enhetstester:
 - En testklass för varje klass i lösningen
 - Varje metod testas – flera testmetoder för varje (många testfall), beskrivande metodnamn
- Exekvera *hela testsviten* vid varje förändring av eller tillägg i enheten
 - ```
public void withdraw(double amount) { // tests?
 if (amount <= balance) {
 ...
 "förbättras" till
 public void withdraw(double amount) {
 if (amount <= balance && amount >= 0) { // test amount < 0
 ...
```
- Vid refaktorisering – kör hela testsviten



# Vad är ett lyckat test?

- Ett lyckat test är ett test som exponerar fel(!), dvs. "falerar"
  - Varför testar vi?
- Ett test som inte exponerar några fel är inte nödvändigtvis misslyckat
  - Men, fundera över om testet täcker alla fall som bör testas

# JUnit

- Ett ramverk för att förenkla arbetet med enhetstester
- CUnit, CPPUnit, Junit, ...
- Integrerat med många utvecklingsmiljöer, t.ex. NetBeans
- *Introduktionsvideo på kurswebben/Övningar*
- Mer? <http://junit.org/junit4/>

# Testdriven utveckling, TDD

- Skriv testen först
- Black box
- Strikt TDD
  - Ingen ny implementation skrivs förrän vi har ett test som visar att existerande implementation inte fungerar
    - Skriv först ett enkelt test, före någon implementation
    - Skriv (bara) implementation så att detta test klaras
    - Lägg till ett nytt test som implementationen inte klarar (osv.)
- För den intresserade:  
<https://www.youtube.com/watch?v=k2tvGCYlp9c>  
<https://www.youtube.com/watch?v=mng8aYoTavs>