TDA550 HT18

# Allmänna OOP-begrepp och Java-tekniker
# Designprinciper
# Designmönster

*Detta dokument innehåller en uppräkning avmaterial som behandlats i kursen. Avsikten är att ge en snabb överblick över moment att repetera. Referenser finns till föreläsningar och övningar samt till avsnitt i kursboken. Listan är inte fullständig. Förslag till förbättringar mottas tacksamt!*

# Diverse osorterade begrepp, tekniker, Java

| Moment | Förel. | Övn. |
|---|---|---|
| Kvalitetskriterier | 1 | |
| Referens, alias | 3 | 2.1, 2.2 |
| Typ och subtyp | 3 | |
| Äkta subtyp | 5 | |
| Typkompatibilitet och typomvandling | 3 | |
| Kovarians | 8 | |
| Modellering och UML | 3 | |
| Testning | 11 | |
| Refaktorering | 11 | |
| Felhantering, undantag | 2, 11 | 6.1 |
| Fail fast, assertion | 11 | |
| Sidoeffekt | 6 | |
| Grund och djup kopiering, clone | 8 | 4.5 |
| Likhet, hashkoder, komparatorer | 8 | 4.2, 4.3, 4.6 |
| Generiska metoder och klasser | 12 | 6.3, 6.5 |
| Javas API för samlingar och tabeller | 2, 12 | 6.2, 6.5 |
| Nästlade klasser | 13 | 6.6 |
| Filer och strömmar | 13 | |

# Begrepp

Statisk typ och statisk bindning

F4, Skr. 2.3,2.6

*En referensvariabels deklarerade typ är dess statiska typ. Vid statisk bindning bestämmer variabelns statiska typ vilken metod som anropas.*

Överlagring och överskuggning

F4, Skr. 2.9

*Överlagring: Flera metoder (eller konstruktorer) i en klass har samma namn men* olika signaturer. *Överskuggning (omdefinition): En metod i en subklass har* samma signatur *som en metod i en basklass.*

Dynamisk typ och dynamisk bindning

F4, Skr. 2.6, Övn. 2.4

*En referensvariabels dynamiska typ bestäms av typen hos det utpekade objektet. Vid dynamisk bindning bestämmer variabelns dynamiska typ vilken metod som anropas.*

Polymorfism (metodpolymorfism)

F4, Skr. 2.6,2.8, Övn. 2.4

*Olika överkuggande metoder med samma signatur anropas med dynamisk bindning i en samling subklassobjekt.*

## Koppling

F1,F5, Skr. 5.7

*Om en klass använder en annan klass har klasserna en inbördes koppling. Koppling kan vara enkel- eller dubbelriktad. Koppling innebär beroenden och komplicerar förändring.*

## Kohesion

F1, Skr. 4.3,5.2

*Graden av självständighet hos en metod, klass eller modul. Hög kohesion innebär att vara väl fokuserad på sin uppgift. Låg kohesion är ofta förknippad med hög koppling och vice versa.*

## Inkapsling

F3,F4, Skr. 5.7

*Att definiera data och tillhörande funktioner (metoder) tillsammans i en klass. Se även Informationsgömning.*

## Muterbarhet

F7, Skr. 5.6, Övn. 4.1

*En klass är muterbar om objekt av klassen kan ändra tillstånd under objektets livstid. En omuterbar klass har konstanta oföränderliga objekt, t.ex. klassen String.*

## Specifikationsarv

F2,F3,F5, Skr. 2.6,2.7

*Specifikationsarv innebär att en klass ärver från (implementerar) ett eller flera gränssnitt.*

## Implementationsarv

F2,F3,F5, Skr. 2.5

*Implementationsarv innebär att en klass ärver data och implementerade metoder från en basklass. I Java tillåts denna typ av arv från högst en basklass.*

## Delegering

F3,F5,F7, Skr. 5.6

*Delegering innebär att en metod tar hjälp av en annan metod (ev. i ett annat objekt) för att utföra en uppgift. Delegering är en generell teknik för att ersätta onödiga implementationsarv.*

## Klassinvariant

F6, Skr. 4.4, Övn. 3.3,4.1

*En klassinvariant är en egenskap hos ett objekts tillstånd (dess instansvariabler) som alltid gäller från det att en konstruktor exekverat klart och vid slutet av varje metodexekvering.*

## Förvillkor

F6, Skr. 4.6, Övn. 2.2-2.5,4.1

*Ett förvillkor i en metod uttrycker egenskaper som klienten som anropar metoden måste uppfylla för att metoden skall kunna utföra sin uppgift.*

## Eftervillkor

F6, Skr. 4.6, Övn. 2.2-2.5,4.1

*Ett eftervillkor i en metod uttrycker egenskaper hos resultatet som metoden åtar sig att uppfylla under förutsättning att förvillkoren är uppfyllda.*

## Kontrakt

F6, Övn. 2.2-2.5,4.1

*Ett formellt avtal om hur en klient får använda en metod. Kontraktet består av för- och eftervillkor för metoden.*

## Specifikation

F6, Övn. 2.2-2.5,4.1

*Kontraktet för en metod utgör dess specifikation.*

# Designprinciper

## Liskov Substitution Principle

F5, Skr. 3.3

*Klass B kan vara en subklass till klass A, eller implementera gränssnittet A, endast om för varje metod som finns i både A och B:s gränssnitt, B:s metod accepterar alla de värden som A:s metod accepterar (och eventuellt mer) och gör allt med dessa värden som A:s metod gör (och möjligen mer).*

## Principle of Least Astonishment

F5, Skr. 3.3

*Om man i en klient byter ett objekt av typ T mot ett objekt av en subtyp till T så får inga oväntade saker hända. Speciellt får det ersättande objektet inte kasta oväntade undantag.*

## Dependency Inversion Principle (DIP)

F4,F5, Skr. 5.7

*Programmera mot gränssnitt, inte mot klasser. Ett viktigt exempel är programmering med iteratorer. Koden beror av gränssnittet Iterator, inte av de konkreta iteratorklasserna (som är anonyma).*

## Open-Closed Principle (OCP)

F5, Skr. 5.7

*Klasser skall vara öppna för tillägg men slutna för förändring. Det betyder att man skall kunna lägga till ny funktionalitet utan att behöva ändra i befintlig kod. Se även DIP och designmönstret Strategy.*

## Uniform Access Principle

F7, Skr. 5.5

*Information skall kunna nås på ett enhetligt sätt, via metoder med beskrivande namn. Java bryter ibland mot denna princip. Metoden size ger antalet element i en lista men attributet length storleken hos ett fält.*

## Interface Segregation Principle

F7

*Gränssnitt skall anpassas till klienters behov och inte innehålla överflödiga metoder som saknar relevans för klienterna.*

## Separation of Concerns Principle

F6, Skr. 4.3,5.3, Övn. 3.1

*En metod skall göra* en *sak och göra det bra. Det skall gå att beskriva metoden i en mening utan att använda orden* och *eller* samt.

## Command-Query Separation Principle

F4,F6, Skr. 4.3

*En metod skall inte både vara en accessmetod och en muterande metod. Jfr. metoden next() i Iterator som avviker från denna princip. Sidoeffekter gör sådana metoder svåra att förstå.*

## Information Hiding Principle

F4, Skr. 2.10

*Data i en klass skall hållas osynligt för klassens klienter. Genom att följa principen undviker man att klienter kan bli beroende av klassens implementeringsdetaljer, vilket eliminerar oönskad koppling.*

## Information Expert Principle

F7, Skr. 5.3,5.4

*Den klass som äger viss information skall vara ansvarig för bearbetning av informationen.*
*Jfr Tell don't ask Principle.*

## Tell don't ask Principle

F7

*En klass skall inte utföra uppgifter åt andra klasser, baserat på deras data, som de istället kan göra själva. Jfr. Information Expert Principle.*

## Law of Demeter: Don't talk to strangers

F7, Skr. 5.8

*En klass bör bara anropa egna metoder och metoder i sina närmaste grannar, inte i mer avlägsna klasser. Följs principen fås lägre koppling.*

# Don’t repeat yourself (DRY)

Skr. 5.4

*… har vi inte redan sagt detta? Nej, tror inte det! Undvik duplicerade kod. Det är en mycket vanlig felkälla eftersom förändringar lätt genomförs inkonsekvent på sådan kod. Undwik dplicerd kod!*

# Designmönster

## Decorator Pattern

F10, Övn. 5.6,5.7

*Strukturellt mönster för att lägga till funktionalitet till objekt. Mönstret bygger på delegering. Många exempel finns i Javas IO-ramverk.*

## Composite Pattern

F10, Skr. 8.7, Övn. 5.5

*Strukturellt mönster för att beskriva trädstrukturer, t.ex. filsystem, hierarkiska bilder, komponent- och behållarhierarkier i grafiska gränssnitt. Jfr Java-klasserna Component och Container i java.awt.*

## Adapter Pattern

F10, Skr. 7.1

*En adapter är en klass som ger klienter ett anpassat operationsgränssnitt till ett objekt genom att delegera klientanrop till anrop på det adapterade objektet.*

## Facade Pattern

F10, Skr. 9.11

*En fasad är en mer generell adapter som ger klienter ett ”skräddarsytt” operationsutbud från ett urval av metoder hos en eller flera klasser.*

## Observer pattern

F10, Skr. 8.2, Övn. 5.4

*Strukturellt mönster som minskar kopplingen mellan modellklasser och observatörsklasser. Kärnan i arkitekturmönstret Model View Control.*

## Model View Control Pattern

F10, Skr. 8.4

*Arkitekturellt mönster som minskar kopplingen mellan modell och vy i applikationer med grafiskt gränssnitt. Innehåller Observer som en väsentlig del.*

## Singleton Pattern

F9, Skr. 7.2, Övn. 5.1

*Programmeringsidiom som beskriver hur en klass skall konstrueras för att garantera att den bara kan instansieras en gång. I Java med klassvariabel av typen och privat konstruktor, alt. enum-konstruktion*

## State Pattern

F9, Skr. 8.6, Övn. 5.2

*Strukturellt mönster för att konstruera utbytbara tillstånd som objekt.*

## Template method Pattern

F9, Övn. 5.7

*Utbytbara metoder i en övergripande algoritm definieras som hook-metoder i subklasser. Ex. integrering av matematisk funktion.*

## Strategy Pattern

F9, Övn. 2.5,5.3

*Låter en metod ta en utbytbar metod som inparameter genom att parametermetoden kapslas in i en funktionsobjektklass. Ex. Klassen TreeSet:s konstruktor kan ta ett komparatorobjekt som parameter.*

## Factory Method Pattern

F4,F10, Skr. 7.5

*En objektfabrik har en metod som tillverkar objekt åt klienter baserat på ett eller flera parameter-värden. Beroenden till de inblandade typnamnen isoleras till fabriken. => Ett ställe att ändra på!*

## Iterator Pattern

F12, Skr. 7.3, Övn. 6.6

*Ger klienter möjlighet att iterera i samlingar på ett syntaktiskt enhetligt sätt oberoende av skillnader i underliggande implementeringstekniker.*
*En tillämpning av Factory Method.*