Viktige begreper (Engelsk – Norsk)

Inheritance – Arv (Kap. 10)

Superclass (or parent class) – superklasse (eller foreldreklasse): Klassen som subklassene arver fra. Inneholder alt det som er felles (felt og metoder) for alle objekter av subklassene. Eks: Post.

Subclass (or child class) – subklasse (eller barneklasse): Klassen som arver fra superklassen. Eks: MessagePost. Kan ha egne felt og metoder i tillegg til de som den arver. I Java bruker vi det reserverte ordet extends etter klassenavnet for å vise at det er en subklasse. Eks: public class MessagePost extends Post.

*Is a* – relationship : Subklassen *er* en spesialisering av superklassen. Dette i motsetning til *has a* – relationship som betyr at an klasse *benytter seg av* en annen klasse.

super()  (I subklassens konstruktør): Kaller på superklassens konstruktør. Kan inneholde parametre. Eks: super(author) kaller på superklassens konstruktør med parameteren author. Husk å kalle på superklassens konstruktør først i subklassens konstruktør.

Subtype – Subtype: Klasser er også typer. En subklasse er en type. En subklasse er også en subtype av typen som superklassen definerer.

Assign – tilordne: Når vi gir en variabel en verdi, så sier vi at vi tilordner den en verdi. Eks: int i = 0; variabelen i tilordnes verdien 0. En variabel må tilordnes en verdi av deklarert type **eller en subtype av denne**.

Polymorph (many shapes) – polymorf (flere former): En variable av en type (eks Post) er polymorfisk fordi den kan tilordnes typen Post eller en subtype av Post (Eks: MessagePost). Den kan altså holde i flere ulike typer objekter.

Substitution – substitusjon: Når vi bruker en subtype der vi har deklarert plass til en supertype. Eks: Post post = new MessagePost();

Cast - cast: Hvis vi vet at et objekt i realiteten er et mer spesialisert objekt (en subtype), så kan vi velge å caste. Det gjøres ved å presisere typen du skal caste til mellom to parenteser. Eks (vi vet at objektet post er av typen MessagePost):

MessagePost messagePost = **(MessagePost)** post;

More about inheritance (ch 11)

static type - statisk type: Statisk betyr noe som ikke forandrer seg. Den statiske typen til en variabel er den typen variabel er deklarert som. Eks: Vehicle v;. Den statiske typen til variabelen er Vehicle.

dynamic type - dynamisk type: Dynamisk er det motsatte av statisk. Da kan altså noe endre eller utvikle seg. Den dynamiske typen til en variabel er den typen variabelen har for øyeblikket. Variabler er jo polymorfiske; de kan holde objekter av forskjellige typer.

overriding - overstyring: En subklasse kan overstyre en metode i superklassen. Metoden må ha samme navn (og resterende signatur) som i superklassen. Metoden i subklassen vil da ha prioritet foran metoden med samme signatur i superklassen. Bruk gjerne annotasjonen @Override over metoden i subklassen for å tydeliggjøre at dette er en metode som overstyrer en annen.

signature - signatur: metodenavn og parametre. Eks: public void display(String s): Signaturen er navnet “display” og en parameter av typen String. private String display(String string) har altså samme signatur.

Further abstraction techniques (kap 12)

Abstract class - abstrakt klasse: En klasse som det **ikke** er tiltenkt å lage instanser av. Den skal fungerer som superklasse for andre klasser. Den inneholder gjerne abstrakte metoder.

Abstract method - abstrakt metode: Metode uten “kropp” (innhold). Innholdet defineres i subklassen. Abstrakte metoder kan kun ligge i abstrakte klasser.

Abstract subclass - abstrakt subklasse: En subklasse som **ikke** implementerer alle abstrakte metoder i superklassen. Subklassen blir abstrakt ettersom den da inneholder minst én abstrakt metode.

Multiple inheritence - multippel arv: Å arve fra flere superklasser. I Java kan en klasse ikke arve fra mer enn én klasse, men den kan *implementere* flere interface…

Interface - grensesnitt (sier gjerne interface på norsk): Spesifikasjon av en type. Har som regel ikke noe implementasjon.

Implement - implemenetere: Iverksette - realisere. Når vi i en klasse beskriver en klasse slik: class ClassName implements InterfaceName, så medfører det at klassen ClassName **må** implemenetere spesifikasjonen vi finner i InterfaceName. Hvis den ikke gjør det, så vil ikke koden kompilere.

Functional interface - funksjonelt grensesnitt: Et interface som kun inneholder beskrivelse av én abstrakt metode i spesifikasjonen. Dette medfører at vi kan implementere grensesnittet via et lambda-uttrykk.

Lambda expression - lambda-uttrykk: En måte å uttrykke en funksjon på. Dette kan gi oss enklere kode. Syntaksen ser slik ut: (<input - eventuelle argumenter>)-><hva funksjonen skal gjøre>. Eks: (x, y) -> x + y;

Building graphical user interfaces (kap 13)

GUI (Graphical User Interface) - Grafisk brukergrensesnitt: Det stedet der brukeren møter programmet. Det inneholder visuelle elementer som forenkler brukerens interaksjon med programmet (knapper, menyer etc). Det er derfor et *grafisk* grensesnitt.

Component - komponent: Et element i brukergrensesnittet (knapp, tekstfelt etc).

Layout - layout: Hvordan komponentene er plassert ut i brukergrensesnittet. Vi bruker egne komponenter (layout komponenter) til dette.

Nested layout - nøstet layout: Noen ganger må vi benytte layout-komponenter inne i andre layout-komponenter for å få til ønsket grensesnitt. Da kaller vi det en nøstet layout.

Stage - scene (som i den fysiske arenaen i et teater): Den ytre rammen i en JavaFX-applikasjon. I JavaFX arver vi fra den abstrakte klassen Application. Da må vi implementere metoden start(Stage primaryStage). Vi har altså alltid en Stage i en applikasjon (program), men vi kan velge å benytte flere.

Scene - scene (som i en spesifikk del av et skuespill. Eks: Scene 1): Der vi tegner opp brukergrensesnittet vårt. Vi kan ha flere Scenes i en Stage, men vi kan bare vise en av gangen.

Scene-graph - Scene-graf: Alle komponenter i en Scene må være koblet sammen i en graf. Denne grafen må ha en rot-komponent.  Komponentene i grafen kalles gjerne noder.

Node - node: Vanlig betegnelse for et element i en struktur innen informatikk. Eks: Noder i en scene-graf.

Event - hendelse: Noe som skjer i brukergrensesnittet. Eks: Brukeren klikker på en knapp. Da fyres det av gårde en hendelse som noen (komponenter) kan lytte etter. Når en hendelse inntreffer har programmet gjerne egne “handlers - behandlere” for å ta seg av hendelsen.

EventHandler - hendelses-behandler(sier gjerne EventHandler på norsk): Kode som tar seg av hendelsen. Denne koden er gjerne samlet i en klasse eller et lambda-uttrykk.

Inner class - indre klasse: En klasse inne i en klasse. Den indre klassen har tilgang til alle instansvariabler og metoder i den omsluttende klassen.

Anonymous inner class - anonym indre klasse: En indre klasse med en spesiell syntaks som kan benyttes for å implementere klassen på en enkel måte. Vi er ikke interessert i å gi den indre klassen et navn. Vi bruker gjerne anonyme indre klasser for å direkte vise implementasjon av de abstrakte metodene i et interface. Dette var mer vanlig før Java 8. Nå er det vanligere å benytte lambda-uttrykk, men det kan vi jo bare når interfacet er funksjonelt.

Handling errors (Kap 14)

Exception - unntak: Et objekt som omhandler et unntak for normal kjøring av en kodesnutt (noe som har skjedd i en metode). Unntaket kastes og kan fanges for å behandles.

Unchecked exception: Et unntak som kompilatoren **ikke** *sjekker*. Skyldes gjerne feil fra programmererens side. Unntak som arver fra RuntimeException. Ikke meningen at vi skal kunne håndtere disse på en hensiktsmessig måte.

Checked exception: Et unntak som kompilatoren *sjekker*. Unntak som arver fra Exception (men ikke fra RuntimeException). Unntak som gjerne ikke skyldes feil fra programmereren. Unntak som vi vil forsøke å håndtere på en hensiktsmessig måte.

Assertion - Påstand: Vi forsikrer oss om at koden fungerer som tiltenkt, ved å undersøke tilstand via assertions. Hvis vår påstand ikke er korrekt, så er det noe rart i koden vår (eller påstanden er gal). Disse kan brukes under utvikling av koden, og fjernes før koden skal i produksjon. Hvis påstanden feiler vil det bli kastet en Assertion-error.

Serialization - serialisering: Når vi skal “overføre” et objekt på en slik måte at det blir klippet i småbiter og senere satt sammen igjen til nøyaktig samme tilstand: Eks: Skrive/lese til/fra fil. Overføre over nett. Får å få til dette i Java, så må objektet implementere interfacet Serializable. Det er et snodig interface da det ikke inneholder noen abstrakte metoder.