Undervisningen i kursen ska behandla följande centrala innehåll:

Grunderna för klasserna, objekt, egenskaper och metoder.

```
Klasser - mall för objekt
Objekt - instans av klass
Egenskaper - Attribut/Variabler. Databärande innehåll som hör till klassen.
Metoder - funktioner som objekt av klassen kan exekvera. Dessa metoder ändrar
oftast på egenskaper/attribut i objektet.
```

Arv, inkapsling och polymorfism.

En klass kan ärva från en annan klass. Den ärver då attribut och metoder från den klassen. Detta är ett sätt att ha specialiserade "barn" klasser med annorlunda funktionalitet, eller för att minska duplicering av kod mellan klasser som gör liknande saker.

Inkapsling betyder att man låter attribut som tillhör en klass vara "inkapslade" så att endast kod i metoder som klassen äger kan komma åt och ändra värdena. Rent simpelt handlar det om att sätta egenskaper/attribut/instansvariabler till private.

Polymorfism är något som tillåter oss att betrakta flera olika typer av klasser som ärver av en föräldraklass som föräldraklassen. Detta är bra om man t.ex. har ett spel med väldigt många olika typer av karaktärer som är av olika klasser men låt säga att de har en gemensam föräldraklass "Character" som har en metod move(). Då tillåter polymofrism oss att behandla alla barnklasser som en Character och vi kan utföra move() på dem.

Skapande av klasser och objekt i ett objektorienterat programspråk utifrån tidigare analys och design.

Detta ska vi börja träna på direkt. Vi kommer skapa CRC (class-responsibility-collaboration) kort för ett givet scenario och skapa dessa klasser.

Användning av klasser och att genom arv förändra beteende hos klasser som ingår i egna och andras klasshierarkier och standardbibliotek.

En grej med arv är att man både kan ärva metoden och använda den precis som den ser ut i föräldraklassen, men man kan också förändra eller lägga till funktionalitet till metoder man ärvt. Man säger att man gör en "override" av funktionaliteten i den ärvda metoden. Detta kan vi göra på egna klasser och för klasser som finns i standardbibliotek.

Generiska klasser och metoder.

Generiska klasser och metoder tillåter att de kan användas med olika typer av datatyper. Vi behöver inte ställa explicita krav på att metoder och attribut ska vara av en specifik typ som String, int, double, list, osv.

Variablers och metoders synlighet och livslängd.

Handlar både om metoder och variabler som är markerade som public/private/protected, avgör om de ska nås utanför eller inom klassen endast. Handlar även om scope. Funktioner och variabler kan definieras inom globala och lokala scopes vilket avgör hur synliga de är och hur länge de "lever". En variabel som definieras lokalt i en metod lever i funktionens scope medans funktionen körs, sedan upphör den att existera.

Stark och svag samt statisk och dynamisk typning.

I statisk typning är typen bunden till variabeln och typer kontrolleras då programmet kompileras, att du inte försökt nyttja en variabel av en typ på ett sätt som variabler av den typen inte kan nyttjas. T.ex. genom att slå ihop en bool med en sträng, eller tilldela en siffra till en variabel av typen lista.

I dynamisk typning är typen bundet till värdet variabeln har. Om värdet byts till en variabel av en annan typ medans programmet kör är det okej. Det är först när du gör något med en variabel som du inte får. t.ex. att slå ihop en bool med en sträng, som du kan få en error.

I starkt typade språk kommer inte variabler av olika typer automagiskt konverteras från en typ till en annan. Strängen "123" kommer inte tolkas som siffran 123 i ett starkt typat språk. Svagt typade språk kan tillåta den här typen av konvertering.

Den bästa beskrivningen: Statisk/Dynamisk typning bestämmer NÄR information om olika typer sker: Vid kompilering eller när programmet körs.

Stark/Svag typning handlar om HUR STRIKT typer särskiljes av språket? T.ex. om det tillåter automagiska (implicit/underförstådda) typkonverteringar.

Typningen kan vara olika stark och svag beroende på hur många olika typer av regler som påtvingas.

De vanligaste exempelen på starkt typade språk är de som har statisk typning och ej tillåter implicita typkonverteringar.

Identifierares synlighet och livslängd.

Ett objekt har aldrig samma identitet som ett annat objekt. Det är något som unikt identifierar en instans av ett objekt. Två instanser kan vara likadana om deras innehåll är likadana. Då implementerar vi ofta en equals funktion för att jämföra objekt av den typen. Men två objekt ska aldrig kunna vara lika med varandra. Två identifierare kan dock identifiera samma objekt.

Konceptuellt handlar detta om variabler vars värden är objekt.

Det valda programspråkets kontrollstrukturer.

typescript, samma som javascript, täckt

Undantagshantering.

try/catch redan täckt i webserverprogrammering

Analys, nedbrytning och modellering av programmeringstekniska problem med lämpligt analysverktyg, till exempel användningsfall.

I princip, CRC kort + beskrivning av de användarfall som crc korten uppfyller. T.ex. om vi ska programmera ett system för en skola gör vi CRC kort för de komponenter som skall implementeras och beskriver användningsfall som vad användare av systemen ska kunna göra, och isåfall vilka klasser användaren kommer interagera med.

Design av lämplig lösning ur föregående analys med lämpligt verktyg och metoder som klassdiagram.

Klassdiagram är en specifik representation av vår modell. Vi kommer kolla på UML. Det handlar om en visuell beskrivning av klasser, dess metoder, egenskaper och hur de förhåller sig till varandra med pilar.

Skapande av användarvänliga gränssnitt.

Övat på i webbserverprogrammering. Vi över vidare.

Skrivning och läsning av lagrad data.

Övat på i webbserverprogrammering. Vi över vidare.

Utveckling av program som nyttjar kommunikation över internet.

Övat på i webbserverprogrammering. Vi över vidare.