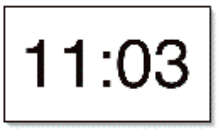
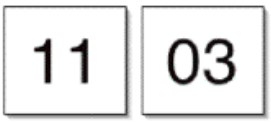
AlarmClock

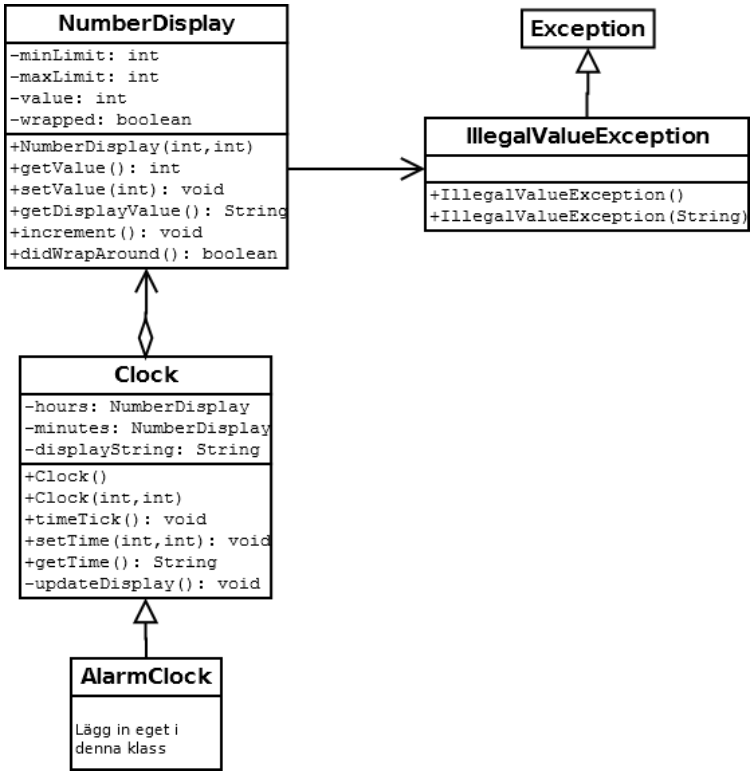
Du ska skapa en digital klocka `Clock`, som kan visa tiden i 24-timmars format. En central del av klockan är displayen som visar tiden. Ett alternativ till att lösa problemet är att hantera allt i en enda klass där displayen består av alla fyra siffror:



Det kommer dock att visa sig att problemet blir för komplext med den ansatsen. Det är bättre att försöka finna ett sätt att dela upp problemet i mindre delproblem som sedan kan sammanfogas till en lösning. En första ansats till uppdelning skulle kunna vara att displayen ska bestå av fyra siffror (två för timmar och två för minuter). Det skulle dock bli en allt för finmaskig indelning. Den display du ska implementera ska istället bestå av två sifferdisplayer, `NumberDisplay`, för timmar och minuter:



Klassdiagrammet för ditt system kommer till en början att se ut så här:



NumberDisplay

Börja med att implementera klassen `NumberDisplay`. Klassen ska symbolisera en digital display som kan visa siffror mellan en nedre och en övre gräns. Dessa gränser (`minLimit` och `maxLimit`) anges som parametrar till konstruktorn när man skapar ett objekt. Värdet (`value`) kan sedan variera mellan `minLimit` och `maxLimit-1`. D.v.s. om `maxLimit` är 60 och `minLimit` är 0, så varierar värdet mellan 0 och 59. När värdet ökas så att det kommer över gränsen ska det automatiskt starta om från `minLimit`.

Metoderna:	**Förklaring:**
`NumberDisplay(int minLimit, int maxLimit)`	Skapar en ny numberdisplay med värdet `minLimit` och gränserna `minLimit` och `maxLimit`. Om `maxLimit` inte är större än `minLimit` ska ett undantag genereras (`IllegalValueException`).
`getValue()`	Returnerar det aktuella värdet på displayen som ett heltal.
`setValue(int newValue)`	Sätter värdet på displayen till `newValue`. Om `newValue` är mindre än `minLimit` eller större än värdet displayen som mest får ha så ska ett undantag genereras (`IllegalValueException`).
`getDisplayValue()`	Returnerar det aktuella värdet på displayen som en sträng. Tänk på att göra strängen "snygg" så att alla värden anges med lika många siffror som det största värdet displayen får ha. Värden som består av färre siffror inleds med ett antal 0:or så att de blir lika långa.
`increment()`	Ökar displayens värde med 1 och ser till att värdet blir `minLimit` om maxgränsen är nådd.
`didWrapAround()`	Ska returnera `true` om displayen just kommit över `maxLimit` och startat om från `minLimit`, annars `false`. Använd attributet `wrapped` för att hålla reda på detta.

Testa din implementation genom att göra en testklass. Börja med att fundera på vilka fel som skulle kunna uppkomma (slår värdet verkligen om, blir strängen "snygg", etc). Skriv även ut i testklassen vad som förväntas.

`IllegalValueException`

Skapa en egen undantagsklass som ärver ifrån `Exception`. Denna ska sedan kastas i klassen `NumberDisplay` när felaktiga värden dyker upp. Skicka med ett felmeddelande när undantag kastas.

Klassen ska ha två konstruktorer. En som tar ett felmeddelande som parameter och en som inte har några parametrar.

`Clock`

När du övertygat dig om att klassen `NumberDisplay` fungerar som den ska så är det dags att implementera `Clock`. Klassen ska symbolisera en digital klocka som kan visa tidpunkter mellan 00:00 och 23:59. Klockan ökar tiden den visar med en minut via `timeTick`-metoden. Timvärdet (`hours`) ökar med ett när minutvärdet (`minutes`) slår om till 0. Attributet `displayString` är en sträng som får simulera en riktig display eftersom vi inte kommer att lägga in denna mjukvara i en fysisk klocka. Undantag som kommer till klassen ska inte tas hand om med try/catch utan kastas vidare.

Metoderna:	**Förklaring:**
`Clock ()`	Skapar en klocka med tiden satt till 00:00.
`Clock (int hour, int minute)`	Skapar en klocka med tiden satt till den tid som ges av parametrarna `hour` och `minute`.
`timeTick()`	Den här metoden är tänkt att anropas varje minut. Den ökar klockans tidsvärde med en minut.
`setTime ( int hour, int minute)`	Sätter klockans tid till den tid som ges av parametrarna `hour` och `minute`.
`getTime()`	Returnerar värdet på `displayString` som en sträng.
`updateDisplay()`	Denna privata metod används för att uppdatera värdet på `displayString` som simulerar en fysisk display.

Testa din implementation genom att göra en testklass som skapar ett Clock-objekt och sedan skriver ut displayens värde, ökar tiden, skriver ut igen etc. Programmet ska testa din klass så pass bra så att du är övertygad om att din implementation fungerar!

Om du vill göra något utöver det specificerade så måste du kolla med läraren ifall det är ok, annars bör du följa det som är specificerat i uppgiften.

Alarm

Ni ska även lägga till alarm till er klocka. Gör en klass `AlarmClock` som **ärver** från `Clock`. Klassen AlarmClock ska alltså INTE innehålla en klocka i attributen, utan vara en klocka.

För att mer funktionalitet senare ska kunna läggas till till larmet kan själva larmfunktionen skötas av en separat klass. Då larmet ska larma så räcker det att ni skriver ut texten "alarm". Om en larmtid är ställd ska klockan larma när klockan nått den tidpunkten. Därefter tas larmet bort. Flera alarm ska kunna vara inlagda samtidigt.

Ni får själva lista ut vilka ytterligare metoder/attribut som behövs för alarm funktionaliteten.

Skriv en testklass som testar de olika funktionaliteterna hos din färdiga alarmklocka.

Redovisning

Uppgiften ska lösas **enskilt** och redovisas muntligt för läraren samt lämnas in på v-klass. Du ska visa upp:

- Testkörningar - vad du testat, varför du testat just det samt vad resultatet blev. Kontrollera noggrant så att dina tester ger förväntat resultat, och kommentera ev. avvikelser från det förväntade.
  Du kommer även att få köra några testklasser som läraren har gjort.
- Kommenterad källkod. Kommentera de delar av koden som kan vara svåra att förstå.
- Javadoc kommentarer ska finnas i koden och genererade websidor ska visas upp.