2D1311 Programmeringsteknik med PBL för S1
Laborationer läsåret 2003/2004

Fyll i ditt namn och personnummer med bläck eller motsvarande. OBS: Om inte denna kvittenssida tas med vid provet och redovisningen får du ingen kvittens (resultatet rapporteras dock in i rapporteringssystemet på Nada).

Kursledare är Olle Bälter, balter@nada.kth.se.

Namn ..... Personnr .....

Laborationer

Laboration 2	Godkänt den ..... (2004−01−27)	Kvitteras	Timmar .....
Laboration 3	Godkänt den ..... (2004−02−03)	Kvitteras	Timmar .....
Laboration 4	Godkänt den ..... (2004−02−10)	Kvitteras	Timmar .....
Laboration 5	Godkänt den ..... (2004−02−17)	Kvitteras	Timmar .....
Skriftligt prov	Godkänt den ..... (2004−02−24)	Kvitteras	Timmar .....

J-del

Spec	Godkänt den ..... (2004−03−01)	Kvitteras	Timmar .....
Granskning	Godkänt den ..... (2004−05−18)	Kvitteras	Timmar .....
Redovisning	Godkänt den ..... (2004−05−18)	Kvitteras	Timmar .....

Laboration 1: Introduktion till datormiljön

Nyckelord: *katalog, katalogträd, rot, fil, källkod, kompilering, avlusning, program, exekvering, tolkning, prototypprogrammering, textbaserat användargränssnitt, inmatning, utmatning, algoritm, paket.*

Mål: *Att du efter laborationen ska klara dig på egen hand i datorsalarna, ha registrerat dig på kursen och förstå uppbyggnaden av ett enkelt Javaprogram. Att du ska förstå innebörden av varje nyckelord och förberedelsefrågor samt kunna ge beskrivande exempel.*

Föreberedelse: *Hitta en jämnbra labbkompis.*

Sammanfattning av labben

I denna laboration ska du orientera dig lite med de program du kommer använda samt skriva Javakällkod till ett litet program. Du ska:

- Titta på kurshemsidan och anmäla dig till kursen samt kopiera filer från kurshemsidan.
- Fixa labbkataloger med lämpliga rättigheter samt förenkla åtkomsten av den gemensamma labbkatalogen.
- Skriva källkoden till ett litet Javaprogram på ett sätt som underlättar skrivandet av större program.

Inloggning

Slå dig ned vid en ledig dator tillsammans med en labbkompis. Ni bör båda ha kvitterat ut var sitt användarkonto (användarnamn och tillhörande lösenord). Välj ett av kontona och logga in.

Webbläsare och kurshemsidan

Med en webbläsare som t ex Internet Explorer kan du utforska Internet där bland annat kurshemsidan finns. Den webbläsaren kan startas genom att från startmenyn välja Internet Explorer. (I labblydelsen skriver vi stegvisa val separerade med '→': `Start` → `Internet Explorer`). Starta webbläsaren och skriv in adressen

`http://www.nada.kth.se/kurser/kth/2D1311/03-04/S/`

samt tryck på returtangenten. På hemsidan finns massor av nyttig information, och du gör klokt i att alltid ha en webbläsare igång och kurshemsidan lättillgänglig när du labbar.

För att slippa skriva in den långa adressen varje gång du ska titta på kurshemsidan bör du spara ett bokmärke (”bookmark”, ”favorite”). Titta bland knappar och menyer för att lista ut hur du ska göra detta!

Registrering till kursen

På kurshemsidan finns information om registrering. Se till att båda blir registrerade. Var noggrann med alla personuppgifter!

Filhanteraren Windows Explorer

Med Windows Explorer kan du utforska datorns och nätverkets *kataloger*. En katalogs innehåll kan bestå av andra kataloger och *filer*. Kataloger som ligger i en annan katalog kallas ibland *underkataloger*.

Kataloger med underkataloger bildar en grenliknande struktur där varje gren är en katalog. Denna grenstruktur kallar man ofta *katalogträd*. Till skillnad från verkliga träd brukar man dock låta katalogträd breda ut sig nedåt och till höger istället för rakt upp. En bieffekt av detta är att katalogträdets *rot* hamnar längst upp till vänster samt att man pratar om att ”gå ned” i en katalog (eller underkatalog).

På en PC finns också något som kallas *enhetsbeteckningar*. Dessa betecknar välanvända enheter så som delar av en speciell hårddisk (både på den aktuella datorn och på en större s k *”server”-dator*), CD-läsare eller diskettstation. Enhetsbeteckningarna består av en bokstav följt av ett kolon, t ex ”`C:`” eller ”`H:`”. En finess med enhetsbeteckningar är att man kan knyta en enhetsbeteckning till en speciell katalog och därmed låta den enhetsbeteckningen utgöra en rot för det underliggande katalogträdet.

En fullständig beskrivning av den plats en fil ligger på brukar kallas *sökväg* och kan till exempel se ut på följande sätt:

```
H:\prg\labbar\introlabb\Test.java
L:\introlabb\Test.java
```

Notera att enhetsbeteckningar, katalognamn och filnamn separeras av ett ’\’.

Starta Windows Explorer och titta runt

Starta Windows Explorer (`Start` → `Windows Explorer`). Windows Explorer är indelad i två halvor. Den vänstra halvan visar katalogträdet med aktuell katalog markerad. Den högra halvan visar innehållet i den aktuella katalogen.

Klicka dig runt lite och se vad som finns. Välj enhetsbeteckningen `H:` och se vad som finns där. Denna katalog är hemkatalogen för den inloggade.

Titta också under `G:`. Detta är katalogträdet för de distribuerade katalogerna i nätverket. Här kan man hitta de flesta hemkataloger, inklusive din egen. Leta reda på hemkatalogerna till båda i labbgruppen. Notera att de filer som eventuellt syns inte automatiskt är läsbara för alla. Det beror helt och hållet på vilka *rättigheter* som är inställda för filen (och katalogen filen ligger i).

Ett klick på höger musknapp ger ofta en meny med ytterligare möjligheter. Detta kan utnyttjas om man till exempel vill skapa en underkatalog eller ändra rättigheter på en katalog eller fil.

Kopiera filer från kurshemsidan

På kurshemsidan finns länkar till föreläsningsanteckningar och Javaexempel. Du ska nu kopiera filerna `startup.bat` och `_emacs` från kurshemsidan.

I webbläsaren kan höger musknapp användas för att få upp en meny med alternativa kommandon. Klicka med höger musknapp på länken `startup.bat`

och välj ”`Save Target As`” (d v s `högerklick` → `Save Target As`). I den dialog som dyker upp ser du till att spara filen på din hemkatalog och med samma namn (`startup.bat`). **Observera** att ”`Save as type:`” skall vara ”`All Files`”, ingenting annat. Spara även filen `_emacs` på samma sätt. **Observera** här att om filen ska kunna vara utan ”efternamn” (`.bat`, `.java`, `.txt`, ...), så måste du ange filnamnet inom citationstecken: `"_emacs"`.

I och med ovanstående manövrar ska du ha ytterligare två filer i din hemkatalog: `startup.bat` och `_emacs` (kontrollera detta!). Du kommer utnyttja dessa filer lite senare i labben.

Att kopiera filer från hemsidan underlättar labbandet väsentligt och du kommer utnyttja denna finess under hela kursen. Kom ihåg att alltid låta ”`Save as type:`” vara ”`All Files`”.

Redigeringsprogrammet Emacs

Starta redigeringsprogrammet Emacs (`Start` → `Programs` → `Emacs`). Redigeringsprogrammet används för att redigera textfiler och Emacs har flertalet finesser om man skall skriva *Javakällkod*. En av de viktigaste är *indentering*. Med indentering menas att texten i Javakällkoden skjuts in en bit beroende på vilken del av programmet som texten utgör.

Filer och buffertar

Två viktiga begrepp i Emacs är *fil* och *buffert* (”file”, ”buffer”). En fil är något som finns sparad på en hårddisk. Om datorn slås av kommer en fil finnas kvar och kan utnyttjas när datorn startas igen. En buffert är något som används tillfälligt under tiden du skriver. Ändringarna i en buffert finns inte automatiskt kvar om datorn slås av. **Det är alltså viktigt** att spara ändringar i en buffert till en fil lite då och då. Detta gäller speciellt om filen ska utnyttjas till något annat (t ex kompilering).

Öppna en fil

Öppna den befintliga filen `startup.bat`. Detta kan göras via menyn (`Files` → `Open File`) eller genom tangentkombinationer (`C-x C-f`, d v s håll ned kontrolltangenten, tryck ’`x`’, håll ned kontrolltangenten, tryck ’`f`’). Längst ned i Emacs dyker då en rad med den aktuella sökvägen upp. Ersätt den aktuella sökvägen med `H:/startup.bat`. Notera hur Emacs använder ’`/`’ för att separera katalognamn och filnamn, precis som i Internetadresser. **Tips!** Med `TAB`-tangenten kompletterar Emacs så långt det är möjligt namnet på den katalog eller fil ni skrivit in, vilket snabbar upp valet av en fil väsentligt (detta kallas ”TAB completion” på engelska).

När hela sökvägen är inskriven trycker du på returtangenten för att bekräfta valet. Innehållet i filen `startup.bat` ska nu dyka upp i en av Emacs buffertar och börja enligt:

```
rem  Rader som börjar med "rem" är kommentarer ("remarks")
rem  Log för denna fil:
...
```

Observera att om den valda filen inte existerar, så kommer Emacs öppna en ny fil med det valda namnet. Med andra ord används med fördel `Files` → `Open File` eller `C-x C-f` både för att öppna en befintlig fil och för att skapa en ny!

Redigering och användbara kommandon

Så fort du redigerar lite i en buffert i Emacs så kommer buffertens innehåll inte stämma överens med den sparade filens innehåll. Detta visas i Emacs genom att markeringen "`**`" visas till vänster om filnamnet längst ned. Sparas bufferten i en fil så försvinner markeringen.

Börja med att spara den aktuella bufferten i en fil med namnet `slask.bat`. Att spara till en fil med annat namn görs med `Files` → `Save Buffer As` eller `C-x C-w`. **Var noggrann** med STORA och små bokstäver!

Ändra ordet "`u1xxxxxx`" i bufferten till "`u1yyyyyy`". Notera hur markeringen "`**`" dyker upp. Spara bufferten (`Files` → `Save Buffer` eller `C-x C-s`) och notera hur markeringen "`**`" försvinner.

Du har nu använt några av de vanligaste kommandona i Emacs. Det finns otroligt många fler, men för denna kurs kan nedanstående kommandon vara bra att komma ihåg. Med "`C-`" avses kontrolltangenten nedtryckt, med "`M-`" avses "meta"-tangenten nedtryckt. På en PC är `Esc`-tangenten samma som meta-tangenten.

Funktion	**Menyval**	**Kommando**
Avbryt		`C-g`
Öppna befintlig fil/ skapa ny fil	`Files` → `Open File...`	`C-x C-f`
Spara buffert i fil	`Files` → `Save Buffer`	`C-x C-s`
Spara i fil under annat namn	`Files` → `Save Buffer As...`	`C-x C-w`
Stäng buffert	`Files` → `Kill Current Buffer`	`C-x k`
Ångra	`Edit` → `Undo`	`C-_`
Klipp ut från markören till slutet av raden (kan upprepas)		`C-k`
Start av markering		`C-mellanslag`
Slut av markering (kopiera)	`Edit` → `Copy`	`M-w`
Slut av markering (klipp ut)	`Edit` → `Cut`	`C-w`
Klistra in	`Edit` → `Paste`	`C-y`
Indentera Javakällkod på en rad		`TAB`
Skriv ut aktuell buffert	`Tools` → `Print` → `Print Buffer`	
Sök	`Search` → `Search...`	`C-s`
Sök och byt	`Search` → `Query Replace...`	`M-%`

Ordna labbkataloger

Du ska nu skapa och sätta passande katalogrättigheter på de kataloger som du kommer använda under kursen. Läs noga beskrivningen som följer, och hoppa inte över något steg! **Tips!** Om en av er läser och den andra sköter tangentbord och mus, så går det lättare!

Visa med Windows Explorer innehållet i den inloggades hemkatalog (via `H:`). Skapa en katalog med namnet `prg` (som är en förkortning av "program-

meringsteknik"). Gå ned i denna katalog och skapa en katalog med namnet `labbar`.

Katalogen `labbar` kommer bli en gemensam labbkatalog. Notera att den än så länge bara är tillgänglig via den inloggade personens konto.

Ändra katalogrättigheter

Ändra rättigheterna för katalogen `labbar` genom att markera den och välja `högerklick → Properties`. Välj fliken "`Security`". Tryck på "`Add`" för att lägga till en person till åtkomstlistan. Notera att ett fönster med två delar dyker upp. Den övre delen innehåller en lista på alla användarkonton du kan lägga till. Då listan är jättelång bör du undvika att hoppa runt i den. I den nedre delen av fönstret kan du skriva in delar av ett användarkonto och få upp förslag. Ersätt all text i det nedre fönstret med labbkompisens kontonamn och klicka på "`Check Names`". Notera hur det fullständiga kontonamnet dyker upp. Klicka på "`Ok`" för att lägga till personen i åtkomstlistan.

Nu finns labbkompisens användarkonto med på åtkomstlistan för katalogen, men *vilka* rättigheter den användaren har är fortfarande inte ändrat. Markera alltså användarkontot för labbkompisen och kryssa i alla "`Allow`"-rättigheter utom "`Full Control`" och klicka på "`Ok`". Nu ska både den inloggade och den nytillagda i gruppen ha tillgång till labbkatalogen. Notera att du kan ge fler labbkompisar tillgång till katalogen genom att upprepa förfarandet för deras användarkonton.

Knyt enhetsbeteckning till labbkatalogen

För att förenkla hanteringen av era labbfiler ska samtliga i labbgruppen knyta en enhetsbeteckning (`L:`) till den gemensamma katalogen `labbar`. Det är då viktigt att komma ihåg vilken användare som skapade och ändrade rättigheterna för katalogen. Vilken av er har den gemensamma katalogen?

Nedanstående ska genomföras av varje labbgruppsmedlem som ska ha tillgång till den gemensamma katalogen.

Du som vill fixa din `L:` loggar först in och kopierar filerna `startup.bat` och `_emacs` från kurshemsidan till din hemkatalog med hjälp av Internet Explorer och "`Save Target As`". Öppna sedan filen `H:/startup.bat` med Emacs och läs noga kommentarerna. Det enda som behöver ändras i filen är vilken användare "`net use`"-raden hänvisar till. **Var noga** med att få användarnamnet rätt.

Spara filen och avsluta emacs. Öppna Windows Explorer och leta rätt på filen du just ändrade (`H:/startup.bat`). Dra nu denna fil till `Start → Programs → Startup → ...` där du släpper den. Kontrollera att `Start → Programs → Startup → Shortcut to startup.bat` existerar. Om inte, så måste du dra dit filen igen.

Logga nu ut och sedan in igen för att med Windows Explorer kontrollera att du har en enhetsbeteckning `L:`. Har du inte det är det dags att kontrollera anvisningarna steg för steg i denna lydelse från början.

När ni båda är nöjda med era `L:` kan du gå vidare med labben.

Kommandofönstret

Kommandofönstret (DOS-fönstret) används för att *kompilera* Javakällkod och exekvera program. Man kan göra mycket mer i detta fönster, men detta är

i huvudsak vad ni kommer göra. Observera att alla kommandon i kommandofönstret måste bekräftas med returtangenten. **Tips!** "TAB completion" kan utnyttjas i kommandofönstret!

Starta kommandofönstret (`Start` → `Command Prompt`). Byt aktuell enhet till `L:` genom att skriva "`L:`" och trycka returtangenten.

Skriv "`dir`" (som i "directory") för att lista innehållet i aktuell katalog. Katalogen är förmodligen tom eftersom du inte har lagt till något där ännu. Till denna introduktionslabb är det bra med en katalog `introlabb`. Fixa en sådan! (**Tips!** "Ordna labbkataloger"). Kopiera även filen `Maze.java` från kurshemsidan till `introlabb`-katalogen.

Välj aktuell katalog att vara `L:\introlabb` genom att skriva "`cd introlabb`" (den aktuella enheten är redan `L:`, så det behöver inte anges). "`cd`" står för engelskans "change directory", byt katalog. Kontrollera nu vilken som är den aktuella katalogen med hjälp av `cd` (d v s "`cd`" utan att ange ett katalognamn). Den aktuella katalogen ska vara `L:\introlabb`. Om du vill hoppa upp till katalogen ovanför skriver du "`cd ..`". Ett kortnamn för "katalogen ovanför" är nämligen "..".

Kolla innehållet i `introlabb`-katalogen (med `dir`). Den bör åtminstone innehålla den nyss kopierade filen `Maze.java`.

Kompilera och exekvera

Javakällkoden till ett program måste först kompileras till s k Java-byte-kod för att senare kunna *exekveras* (köras). Att kompilera Javakällkod görs med kommandot `javac`, där 'c' står för "compiler". Skriv nu "`javac Maze.java`" och tryck på returtangenten. Om filen vore rätt skriven skulle inga fel uppträda och en ny fil med namnet `Test.class` skapas. Eftersom det finns ett fel i programmet får du ett fel liknande:

```
Maze.java:68: cannot resolve symbol
symbol  : variable brown
location: class java.awt.Color
    Color[] color = {Color.brown,
                                 ^
1 error
```

Det betyder att det finns ett fel på rad 68 i filen `Maze.java`. Felet beror på att det inte finns någon fördefinierad färg med namnet `Color.brown` i Java.

Öppna filen i Emacs och rätta till detta genom att byta ut färgen mot `Color.black`. Glöm inte att spara. Försök sedan kompilera programmet på nytt. Nu ska det gå bra! Vid kompileringen skapas flera nya filer som alla slutar med `.class`. Dessa filer innehåller det körbara programmet. Varje fil motsvarar en del (klass) av programmet. Filen `Maze.class` innehåller den klass där programmet startar, d v s den klass som har metoden `main()`. Kontrollera att filerna finns med hjälp av kommandot `dir`!

Det kompilerade programmet (`Maze.class`) kan exekveras med en *Javatolk* som översätter (tolkar) Java-byte-koden till den kod som datorn förstår (*maskinkod*). Exekvera (kör) nu programmet med hjälp av Javatolken: "`java Maze`". Observera att "`.class`" inte skall skrivas ut!

Användbara kommandon

Kommandofönstret har en mängd kommandon och nedan finns exempel på de mest användbara för denna kurs.

Funktion	**Kommandoexempel**
Kontrollera aktuell katalog	`cd`
Byt aktuell enhet (till `L:`)	`L:`
Byt aktuell katalog (till underkatalogen `introlabb`)	`cd introlabb`
Byt aktuell katalog (till "katalogen ovanför")	`cd ..`
Lista innehåll i aktuell katalog	`dir`
Kompilera Javakällkod	`javac Programmet.java`
Exekvera Javaprogram	`java Programmet`

Första Javaprogrammet

Du ska nu skriva ett eget javaprogram `Hej` tillsammans med din labbkompis. Skapa först en ny fil med namnet `Hej.java` och innehållet

```
// Hej.java
// Första Javaprogrammet av <ditt namn>.

public class Hej {
    public static void main(String[] args) {
    }
}
```

I nuläget är programmet det minsta möjliga Javaprogram som går att skriva (undantaget kommentarerna i början). Titta på strukturen. Hela programmet består av en *klassdeklaration* av klassen `Hej` som innehåller en enda *metoddeklaration* av metoden `main()` som i sin tur inte innehåller någon Javakällkod. Lägg speciellt märke till att allt innanför

```
public class Hej {
```

och

```
}
```

är indenterat (inskjutet). Att indentera på detta sätt är att föredra för att se vilken Javakällkod som hör till vad. Då du trycker på `TAB`-tangenten i Emacs skjuts texten automatiskt in så långt den ska. Om texten hamnar fel är det troligt att någon rad ovanför är felaktig (t ex att ett semikolon eller en parentes saknas). Utnyttja därför indentering med `TAB`-tangenten ofta!

Kompilera och provkör programmet! Som synes händer inte mycket vid exekvering. Anledningen till detta är givetvis att vi inte skrivit vad som skall utföras i vårt program. Notera dock att programmet startar och avslutas. Med andra ord har vi ett komplett Javaprogram (som inte gör någonting synbart)!

Om vi vill lägga till Javakällkod finns det två alternativ. Det ena är utanför klassen `Hej`, det andra är inuti klassen. Inuti klassen har vi möjligheten att lägga något direkt eller inuti `main()`. Ni ska nu lägga till två saker. Den ena inuti `main()` och den andra inuti klassen men utanför `main()`.

Lägg till följande källkod inuti `main()`:

```
System.out.println("Välkommen!");
```

Spara bufferten och kompilera programmet. Om Emacs klagar på svenska tecken när du sparar, acceptera förslaget på teckenuppsättning (”coding system”) genom att trycka på returtangenten. Om det blir fel vid kompileringen, läs noga felmeddelandet och försök rätta! Var noga med STORA och små bokstäver. Om du rättar, se till att alltid spara bufferten igen **innan** du kompilerar på nytt. Att rätta källkod kallas för *avlusning* (från engelskans ”debugging”, som härrör från att luftfiltren till kylsystemet på riktigt gamla datorer måste rengöras från insekter och småkryp, ”bugs”).

När du inte får några kompileringsfel så kan ni exekvera programmet. Vad blir resultatet?

Lägg nu till följande källkod inuti klassen `Hej` men utanför `main()`:

```
static BufferedReader indata =
    new BufferedReader(new InputStreamReader(System.in));

static PrintStream utdata = System.out;
```

Spara bufferten och kompilera programmet. Även om allt är helt korrekt skrivet klagar nu Javakompilatorn bland annat på att den inte känner igen ”`BufferedReader`”. Detta beror på att vi inte sagt till Java var klassen `BufferedReader` finns. Klassen `BufferedReader` och en hel del andra klasser som har med in- och utmatning att göra finns samlade i ett *paket* ”`java.io`”. För att säga till java att ta med klasserna i detta paket, lägg till följande rad överst (d v s utanför klassen `Hej`) i din Javakällkod:

```
import java.io.*;
```

Spara bufferten och kompilera programmet. Rätta eventuella fel innan du fortsätter.

Ändra och komplettera nu källkoden inuti `main()` till följande:

```
utdata.println("Välkommen!");

utdata.print("Vad heter du? ");
String namn = indata.readLine();

utdata.println("Hej " + namn + "!");
```

Spara och kompilera programmet. Denna gång ger Javakompilatorn felmeddelandet:

```
Hej.java:XX: unreported exception java.io.IOException; must
be caught or declared to be thrown
```

Det kompilatorn försöker säga är att det kan uppträda ett fel (”exception”) som antingen måste fångas av programmet (”be caught”) eller sägas till Java att felet **får** uppträda genom att deklarera detta vid mettoddeklarationens början (”declared to be thrown”). För tillfället är det sistnämnda att föredra. Ändra således raden

```
public static void main(String[] args) {
```

till

```
public static void main(String[] args) throws IOException {
```

Du har nu sagt till att `main()` kan ”kasta” felet `IOException` vid exekvering. Spara och kompilera igen. Rätta eventuella fel. Exekvera programmet.

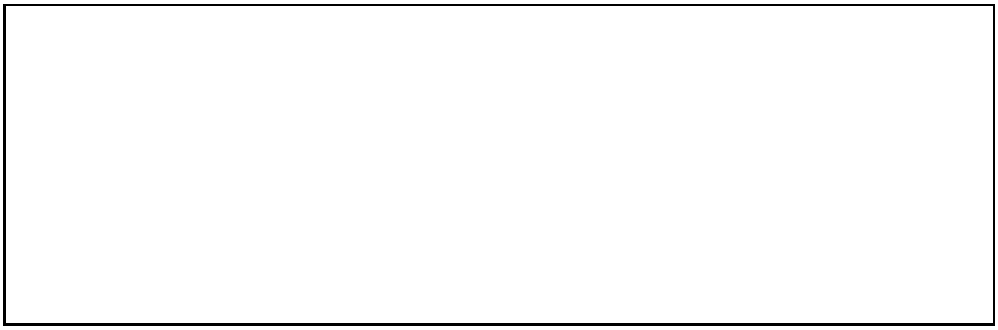
Prova med att svara olika saker på frågan ”Vad heter du?”. Fundera på vilka delar av programmet som ger utskrifterna samt vilka delar som tar hand om det ni skriver på tangentbordet.

Du ska nu låta programmet göra lite mer, nämligen att fråga efter användarens ålder och skriva ut hennes ålder när hon tar examen (om studierna tar fyra år). Börja med att lägga in kommentarer vad som ska göras sist inuti `main()`:

```
// Skriv ut text med fråga om användarens ålder.
// Läs in användarens ålder.
// Omvandla det inmatade till ett heltal.
// Utför beräkningar.
// Skriv ut examensåldern.
```

Kommentarerna/punkterna ovan är en steg–för–steg beskrivning eller *algoritm* över hur programmet ska fungera. Inom programmeringstekniken är det väldigt viktigt att ha algoritmen helt klart för sig innan programmet börjar skrivas; en felaktig algoritm ger garanterat ett felaktigt program!

Titta på punkterna i algoritmen. Vilka variabler behövs för att hålla reda på all information? Är de referensvariabler eller primitiva variabler? Ge variablerna vettiga namn och rita minnet för alla variabler!



Skriv nu den nödvändiga javakällkoden för varje steg i algoritmen (direkt under punktens kommentar). Kompilera och provkör ofta (minst efter varje punkt i algoritmen). Glöm inte att alla variabler måste deklareras innan de används. **Tips!** Man kan få heltalsvärdet av en sträng `enSträng` med hjälp av metoden `parseInt()` i klassen `Integer`:

```
ettHeltal = Integer.parseInt(enSträng);
```

Se till att programmet fungerar som det ska och gå allra sist igenom hela programmet och försäkra dig om att du förstår de rader du lagt till.

Hederskodex

På Nada används en gemensam hederskodex för alla Nadakurser. Du hittar den via kurshemsidan och är själv ansvarig att läsa igenom och följa den.

Vilka är de fem reglerna i hederskodexen?

Sammanfattning av det första Javaprogrammet

Under utvecklingen av Javaprogrammet i denna labb utnyttjas en enkel form av så kallad *prototypprogrammering*. Prototypprogrammering innebär att man utgår från ett enkelt men komplett program och sedan bygger ut det steg för steg med kontroller (så som kompilering och exekvering) efter varje steg. Denna teknik är väldigt kraftfull och används alltid för stora program. För mindre program kan den med fördel användas för att bespara konstiga kompilerings- och exekveringsfel. Kärnan i hela tekniken är att inte göra för stora ändringar mellan kontrollerna. Man skall dessutom också alltid bara ändra i en del av programmet åt gången!

Innan ett program skrivs ska en *algoritm* med de steg som måste finnas med skrivas ned som kommentarer. Ur algoritmen kan man sedan ta reda på vilket minne som behövs och vad man ska kalla variablerna. Med större program kan det ibland vara lättare att först bestämma vilken information som behövs (minnet) och sedan fundera på vad som ska göras (algoritmen).

Det program du precis skrivit har ett helt *textbaserat användargränssnitt*, d v s vid exekvering så består både *utmatningen* (det som programmet skriver ut på skärmen) och *inmatningen* (det som användaren skriver in m h a tangentbordet) av vanlig text.

I Java finns det redan skriven källkod som underlättar för de som skriver nya program. I Javaprogrammet i denna labb utnyttjas bland annat klasserna `BufferedReader` och `PrintStream`. Med lite översättning och fantasi kan du nog redan nu lista ut vad dessa är till för. Klassen `BufferedReader` beskriver en buffrad, d v s mellanlagrande, läsarmanick. Klassen `PrintStream` beskriver en utskriftsströmsmanick, d v s en manick som skriver ut en ström av tecken.

Slutkontroll av laborationen

- ☐ En av oss (nämligen den med användarnamnet .....) har en katalog `H:\prg\labbar` som båda i gruppen har rätt att använda.
- ☐ Jag har knutit enhetsbeteckningen `L:` till ovanstående katalog.
- ☐ Jag har sparat bokmärke(n).
- ☐ Jag har anmält mig till kursen.
- ☐ Jag har hämtat programexempel från kurshemsidan och sparat i labbkatalogen.
- ☐ Jag vet hur jag hittar information på kurshemsidan.

- [ ] Jag har skrivit Javakällkod till ett program, kompilerat, rättat och provkört programmet.
- [ ] Jag har skrivit ned en algoritm, ritat minne över de variabler som utnyttjats av algoritmen och skrivit motsvarande Javakod.
- [ ] Jag vet vad ”prototypprogrammering” innebär.
- [ ] Jag har läst igenom målet med labben och diskuterat eventuella frågor med min labbkompis och/eller handledare.

Laboration 2

Instuderingsfrågorna nedan är indelade efter föreläsningarna. Praktikfrågorna kräver ett program som lösning. Varje vecka ska du dels skriva ett program enligt anvisningarna nedan, dels ska du göra instuderingsfrågorna. *Vid redovisningen ska du kunna förklara både program och svaren till frågorna för dina kurskamrater med hjälp av en skrivtavla.*

Gör följande

Skriv ett program som räknar ut BMI (vikt i kg / (längd i meter * längd i meter)), där en metod används för beräkningen. Experimentera därefter med detta för att besvara följande:

Praktikfrågor

1. Vad är det för skillnad på print() och println()?

2. Vad får man för felmeddelande om man försöker använda en variabel som man inte har deklarerat?

3. Vad får man för felmeddelande om man har fel datatyp på en anropsparameter?

4. Vad får man för felmeddelande om man skriver anropsparametrarna i fel ordning?

5. Vad får man för felmeddelanden om man anropar en metod med fel returtyp (void resp datatyp), parameterantal resp parametertyp samt redovisa hur ovanstående fel uppstår.

Teorifrågor

6. Vad är det för tre saker som kännetecknar en variabel och vad har de för betydelse?

7. Redogör för delarna i metodhuvudet och deras funktion.

8. Vad är parametrar bra till?

9. Varför behöver man anropa metoder?

10. Varför behöver man metoder?

11. Vad är det för skillnad mellan anropsparametrar och formella parametrar?

Laboration 3

Se föreläsning 3: Instanser. Praktikfrågorna kräver ett program som lösning. *Vid redovisningen ska du kunna förklara både program och svaren till frågorna för dina kurskamrater med hjälp av en skrivtavla.*

Gör följande

Skriv en separat klass som hanterar en bil med instansvariablerna körsträcka, tillverkningsår, inköpskostnad, driftskostnad, märke och vikt och skriv instansmetoder för att beräkna årlig körsträcka samt kilometerkostnad. Experimentera därefter med detta för att besvara följande:

Praktikfrågor

12. Hur översätter man en while-slinga till en for-slinga och tvärtom?

13. Visa hur man får felmeddelandet NoClassDefFoundError.

14. Hur får man en NullPointerException och varför uppstår det i praktiken?

Teorifrågor

15. Vad är det för skillnad på en instans och en referens?

16. Vad är det för skillnad på en klass, en metod och en sats?

17. Vad innebär en instansiering?

18. I vilken ordning utförs de olika delarna vid ett anrop av en konstruktor?

19. Vad är det för skillnad på primitiva variabler och String (rita)?

20. Vad är en wrapperklass?

21. När och varför är det bra att rita upp minnet?

22. Hur hanterar Java namnkollisioner?

23. I metodanropet `metod(a);`, vad krävs av parametertypen `a` för att något ska kunna ändra värde? Dvs vilka parametertyper möjliggör att variabler som är nåbara efter den rad där anropet står påverkas av anropet? Rita en minnesbild som förklarar.

Laboration 4

Se föreläsning 4: Instanser och klasser. Praktikfrågorna kräver ett program som lösning. *Vid redovisningen ska du kunna förklara både program och svaren till frågorna för dina kurskamrater med hjälp av en skrivtavla.*

Gör följande

Skriv en klass som hanterar en motorcykel med samma metoder som förra veckans bil, men med en klassmetod för fordonsskatten samt försäkringskostnaden. Skatten är beroende av vikten (75, 160, 210 kg) och försäkringskostnaden av den årliga körsträckan (1000, 1500, 2000 km). Experimentera därefter med programmet för att besvara följande:

Praktikfrågor

24. Vad får man för felmeddelande när man felaktigt försöker anropa en instansmetod som om den vore en klassmetod och tvärtom?

25. Vad får man för felmeddelande när man felaktigt försöker nå en instansvariabel som om den vore en klassvariabel och tvärtom?

26. Visa fördelen med att kunna ha flera instanser av samma klass.

Teorifrågor

27. Vad är this och när existerar den (rita)?

28. Vad kännetecknar en klassvariabel respektive en instansvariabel?

29. Ge exempel på när man bör välja att implementera variabler och metoder som klassvariabler och -metoder respektive instansvariabler och -metoder.

30. Varför behöver man konstruktorer?

Laboration 5

Föreläsning 5: Vektorer. Praktikfrågorna kräver ett program som lösning. *Vid redovisningen ska du kunna förklara både program och svaren till frågorna för dina kurskamrater med hjälp av en skrivtavla.*

Gör följande

Utgående från förra veckans program, lägg till ytterligare en klass som hanterar flera motorcyklar och experimentera för att besvara följande:

Praktikfrågor

31. Vad får man för felmeddelande om man försöker komma åt ett element utanför vektorns gränser?

32. Hur lägger man in element i en hakvektor respektive Vector?

33. Hur byter man ut element i en hakvektor respektive Vector?

34. Hur visar man att inläsningen till en vektor fungerar.

35. Hur anropar man en instansmetod i varje element i en hakvektor respektive Vector?

Teorifrågor

36. Vad är det för skillnad på hakvektorer och Vector?

37. Hur ser strukturen ut när man deklarerar en klass, map instansvariabler, klassvariabler, instansmetoder, klassmetoder och lokala variabler?