Programmeringsuppgift 2

Avsikt

Avsikten med denna programmeringsuppgift och kommande programmeringsuppgifter är att du ska träna på att programmera. Detta innebär att du på egen hand ska lösa samtliga programmeringsuppgifter. Naturligtvis bör du diskutera olika sätt att lösa uppgiften med dina studiekamrater. Men det är viktigt att du själv skriver all din kod och att du inte kopierar från någon kamrat. Det är inte tillåtet att kopiera kod från någon annan.

Programmeringsuppgiften består av ett antal klasser och ett program som ska skrivas. Alla klasser, både de som du skriver själv och de som är givna, ska du placera i paketet **p2**.

Tillsammans med programmeringsuppgiften hittar du filerna **StartRace.java, Car.java, PaintWindow.java, Text.java, TestP2.java och Sound.java**. Du hittar också mappen ***doc***, några bildfiler och jar-filen ***mp3plugin.jar***.

Placera källkodsfilerna i paketet **p2**. **TestP2** används för att testa dina lösningar.

I mappen ***doc*** är det javadoc-dokumentation av klasserna *PaintWindow*, *Sound* och *Text*.

Bildfilerna ***CarBlue.GIF*** och ***CarRed.GIF*** ska du placera i katalogen *images* vilken ska vara i projektkatalogen (tillsammans med *src*-katalogen och *bin*-katalogen). Om du placerar bilderna i någon annan katalog måste du göra motsvarande ändringar i klassen *StartRace*.

mp3plugin.jar ska du placera enligt instruktioner i slutet av uppgiften (endast om du använder ljud i din lösning)

Inlämning

Din lösning av uppgiften lämnas in via Canvas *senast kl 09.00 torsdagen den 18/10*. Du ska placera *samtliga källkodsfiler i paketet p2* i en zip-fil. I zip-filen ska också ett *klassdiagram* finnas (Uppgift 2b).

Zip-filen ska du ge namnet *AAABBBP2.zip* där AAA är de tre första bokstäverna i ditt efternamn och BBB är de tre första bokstäverna i ditt förnamn. Använd endast tecknen a-z när du namnger filen.

- Om Rolf Axelsson ska lämna in sina lösningar ska filen heta AxeRolP2.zip.
- Om Örjan Märla ska lämna in sina lösningar ska filen heta MarOrjP2.zip.
- Är ditt förnamn eller efternamn kortare än tre bokstäver så ta med de bokstäver som är i namnet: Janet Ek lämnar in filen EkJanP2.zip

Granskning

Ca *kl 14.00 den 18/10* kommer en kamrats lösning finnas i din inlämning på Canvas. Din uppgift är att granska kamratens lösningar avseende:

- funktion, design – hur väl uppfyller lösningarna kraven? Är körresultatet korrekt? Hur skulle du utformat klasserna?
- indentering mm, metodnamn, variabelnamn – hur välskriven är källkoden?
- kommentarer – är källkoden väl kommenterad?
- klassdiagram – är diagrammet välritat och korrekt?

Din granskning ska omfatta 1-2 A4-sidor. Granskningen ska vara som pdf-dokument och *lämnas in via Canvas innan du redovisar fredagen den 19/10*. Namnet på granskningen ska vara samma som zip-filen, dvs. *AAABBBP2.pdf*.

Granskning av Programmeringsuppgift 2

Lösning inlämnad av Eva Lind

Granskare: Einar Bok

Datum: 18/10-2018

Funktion, design:
Painting:
…
osv.

Indentering, metodnamn, variabelnamn mm:
….

Kommentarer:
….

Klassdiagram:
…

Redovisning

Redovisning sker *fredagen den 19/10*. Redovisningstid publiceras på Canvas under torsdagen den 19/10. Kom väl förberedd till redovisningen. Kom i god tid till redovisningen så du är beredd då det är din tur. Se till att du är inloggad på en dator (eller har egen dator), att eclipse är igång på datorn och att det går att exekvera dina lösningar. Klassdiagrammet ska vara öppnat i ett program. Din granskning ska också vara öppnad på datorn.

En redovisning sker genom att:

- Studentens lösningar körs med ***TestP2***.
- Granskaren redogör för sina bedömningar
- Studenten svarar för sina lösningar
- Labhandledaren ställer kompletterande frågor

Godkänd uppgift signeras av läraren på lämpligt papper, t.ex. Redovisade uppgifter (se kurssidan). Du ska spara den signerade utskriften tills kursen är avslutad.

Om labhandledaren anser att det endast krävs *mindre komplettering för att lösningen ska godkännas* kan denna komplettering äga rum direkt efter redovisningen. Labhandledaren granskar kompletterad lösning då tiden medger.

Om labhandledaren anser att det endast krävs *mindre komplettering för att granskningen ska godkännas* kan denna komplettering äga rum direkt efter redovisningen. Labhandledaren granskar kompletterad granskning då tiden medger.

Uppgift 2a

Du ska skriva klassen **Painting** (målning) med hjälp av

- Klassdiagrammet till höger
- Nedanstående beskrivning
- Testprogrammet

Painting
- title : String - painter : String - year : int;
+Painting(String, String, int) +getTitle() : String +getPainter() : String +getYear() : int +toString() : String

Beskrivning av klassen Painting

- Klassen Painting är en modell av en målning. Egenskaper i klassen är målningens titel, konstnären och året tavlan färdigställdes.
- När Painting-objektet skapas finns det endast en ***konstruktor*** som går att använda:

  public Painting(String title, String painter, int year) – innehållet i parametrarna förs över till instansvariablerna.
- Med get-metoderna avläser man instansvariablers värde. Det ska vara *en get-metod till varje instansvariabel*.
- Metoden ***toString*** ska returnera en sträng på formen:
  `"TITLE av PAINTER, YEAR"`
  där ord med stor bokstav ersätts med innehållet i motsvarande instansvariabel, t.ex.
  `"Mona Lisa av Leonardo da Vinci, 1507 "`

Testprogram // Placera klassen Exercise2a i paketet p2

```
package p2;
import javax.swing.JOptionPane;

public class Exercise2a {
    public void demo() {
        Painting p1 = new Painting("Mona Lisa", "Leonardo da Vinci", 1507);
        Painting p2 = new Painting("Guernica", "Pablo Picasso", 1937);

        JOptionPane.showMessageDialog(null, p1.getTitle() +
                " målad av den enastående konstnären " +
                p1.getPainter() + ", färdigställd " + p1.getYear());
        JOptionPane.showMessageDialog(null, p2.toString());
    }

    public static void main(String[] args) {
        Exercise2a prog = new Exercise2a();
        prog.demo();
    }
}
```

Exempel på frågor som kan ställas vid redovisningen:

- Vilka instansvariabler innehåller klassen?
- Hur skapar man ett Painting-objekt med egenskaperna: ”Rembrandt”, ”Självporträtt”, 1661
- Hur många Painting-objekt kan man skapa i ett program?
- Vad använder man metoden getPainting() till?
- Vad returnerar toString-metoden?
- Vad gör metoden *JOptionPane.showMessageDialog*?
- Varför måste `import javax.swing.JOptionPane;` finnas i filen?

Uppgift 2b

Du ska skriva klassen **Time** vilken ska innehålla tidsinformation. Du ska också göra ett **klassdiagram** vilket visar klassens namn, instansvariabler, konstruktorer och metoder.

Tiden ska representeras av *timme*, *minut* och *sekund*. Klassen ska alltså innehålla ***instansvariabler*** för att lagra dessa värden. Instansvariablernas typ avgör du själv. Förmodligen kommer du även ha en instansvariabel av typen *Calendar*.

För att erhålla tidsinformation ska du använda klassen *Calendar* vilken finns i paketet *java.util*.

- Ett Calendar-objekt skapas så här:
  `Calendar cal = Calendar.getInstance();`
- Man använder *get*-metoden för att få information från objektet:
  `int minute = cal.get(Calendar.MINUTE);` //
  Fler användabara argument är: `Calendar.HOUR_OF_DAY` och `CALENDAR.SECOND`.
- För att uppdatera ett Calendar-objekt med aktuell tid gör man så här:
  `cal.setTime( new Date() );` // Klassen *Date* finns i *java.util*
  Ett alternativ till att uppdatera befintligt Calendar-objekt är att skapa ett nytt Calendar-objekt.

Vid tidpunkten då objektet skapas ska tidsinformationen initieras. Klassen ska endast innehålla en ***konstruktor***,

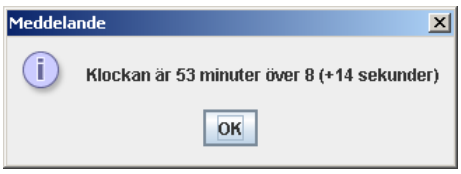
```
public Time() {
    // Skapa Calendar-objekt här
    // Se till att korrekt information lagras i instansvariabler
}
```

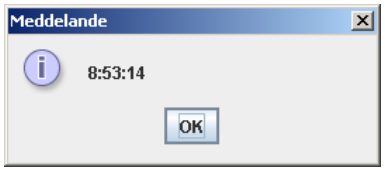
Sedan ska användaren kunna anropa följande ***metoder***:

- **getHour()** – ska returnera timme, **getMinute()** – ska returnera minut, **getSecond()** – ska returnera sekund
- **toString()** – ska returnera en sträng på formen ”TT:MM:SS”, t.ex. 10:16:29 vilket betyder 16 minuter och 29 sekunder över 10.
- **update()** – ska uppdatera den lagrade tidsinformationen. Vid anrop till ovanstående get-metoder ska den uppdaterade tiden returneras.

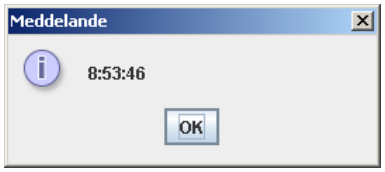
Nedanstående program ska ge ett körresultat liknande figurerna till höger. Programmet demonstrar hur Time-klassen kan användas. Placera klassen Exercise2b i paketet p2.

```
package p2;
import javax.swing.JOptionPane;

public class Exercise2b {
    public void demo() {
        String message1, message2;
        int hour, minute, second;
        Time dt = new Time();

        hour = dt.getHour();
        minute = dt.getMinute();
        second = dt.getSecond();

        message1 = "Klockan är " + minute + " minuter
över " + hour + " (+" + second + " sekunder)";
        message2 = dt.toString();
        JOptionPane.showMessageDialog(null, message1);
        JOptionPane.showMessageDialog(null, message2);
        dt.update();
        message2 = dt.toString();
        JOptionPane.showMessageDialog(null, message2);
    }

    public static void main(String[] args) {
        Exercise2b prog = new Exercise2b();
        prog.demo();
    }
```





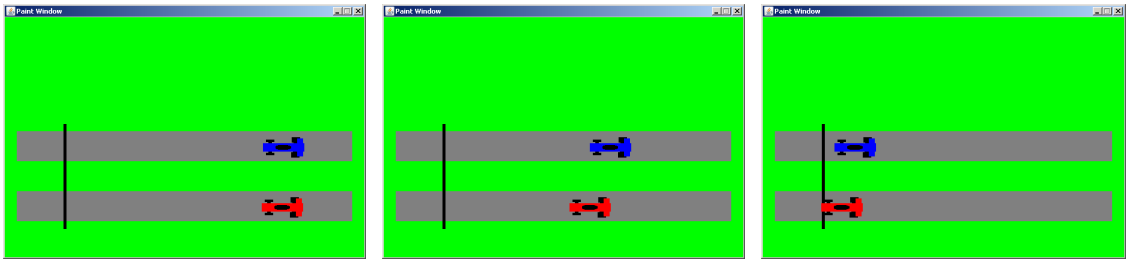
(Anrop till *update*-metoden)



```
}
```

Uppgift 2c

Du ska skriva ett litet program som simulerar en tävling mellan två racerbilar. Racet ska visas i ett **PaintWindow**-objekt.



Krav på spelet:

- ***main***-metoden i klassen ***StartRace*** ska starta spelet. Det är de svartmarkerade raderna som du ska bry dig om. De gråmarkerade raderna (rad 6-9) används av **TestP2** för att stänga PaintWindow-fönstret. De påverkar inte din exekvering.

```
   public static void main(String[] args) {
1      PaintWindow window = new PaintWindow();
2      Car c1 = new Car(new ImageIcon ("images/CarBlue.GIF"));
3      Car c2 = new Car(new ImageIcon("images/CarRed.GIF"));
4      Race race = new Race(window,c1,c2);
5      race.action();
6      if(args.length>0) {
7          PaintWindow.pause(2000);
8          window.dispose();
9      }
   }
```

  I början av *main*-metoden så skapas ett ***PaintWindow***-objekt och två ***Car***-objekt (rad 1-3). Dessa objekt är sedan argument när ***Race***-objektet skapas (rad 4).
  Därför måste klassen *Race* ha en konstruktor som tar emot dessa objekt som argument. Värdena ska i konstruktorn föras över till motsvarande instansvariabler i klassen.

```
public Race(PaintWindow window, Car car1, Car car2) {
    // komplettera med kod
}
```

  Klassdiagrammet till höger visar *nödvändiga* instansvariabler, konstruktorer och metoder enligt beskrivningen ovan.
  Du får lägga till instansvariabler och private-deklarerade metoder i klassen *Race*.

Race
- window : PaintWindow - car1 : Car - car2 : Car
+Race(PaintWindow,Car,Car) +action()

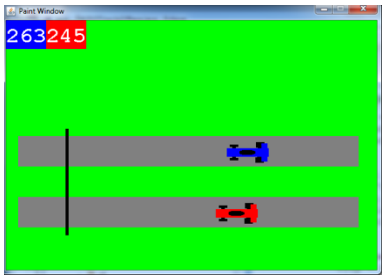
- Spelet startas genom anrop till metoden ***action()***. Metoden action ska i stora drag:
  * Initiera spelet, dvs rita upp bakgrunden, initiera bilarna och rita ut bilarna. Bakgrunden får vara hur elegant som helst.
  * Genomföra ett race. Bilarna ska flytta ett slumpmässigt antal pixlar mot mål varje gång. Racet är slut då en av bilarna nått målet. En while-loop är lämplig för att styra bilarnas flyttningar.

- Klassen ***Car*** är given och ska användas i spelet.
  I klassdiagrammet är *moveTo*-metoden set-metod för instansvariablerna *x* och *y*.

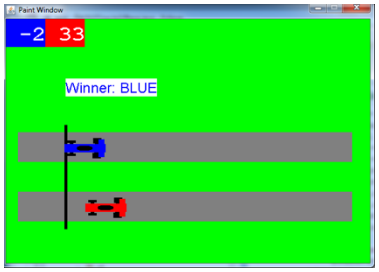
Car
- image : ImageIcon - x : int - y : int
+Car(ImageIcon) +getImage() : ImageIcon +getX() : int +getY() : int +moveTo(int newX,int newY)

Du får gärna göra programmet mer tilltalande genom t.ex.:

- **Visa hur långt vardera bil har till mål (pixlar)**. Figuren visar att blå bil har 263 pixlar till mål och röd bil har 245 pixlar.
  Använd ***Text***-objekt för att visa siffrorna.
  * Det är lämpligt att använda ett Font-objekt med typsnittet ”Monospaced”. Varje tecken har då samma bredd.
  * Använd metoden String.format för att formattera en lämplig sträng att visa i Text-objektet. Text-objektet anpassar sin storlek efter innehållet.



- **Skriva ut vem som vann:**



- **Lägga till lite ljud i programmet (mp3,wav).**
  Skapa katalogerna ***lib*** och ***sound*** i projektet (på samma sätt som du skapade *images*).
  Kopiera ***mp3plugin.jar*** till katalogen *lib*.
  Högerklicka ***mp3plugin.jar*** och välj **Build Path – Add to Build Path**. Nu skapas *Referenced Libraries* och i den ser du *mp3plugin.jar* (dvs. filen kan användas i proejktet)
  Placera filen ***Sound*** i paketet ***p2*** i ditt projekt.
  Placera ljudfiler i katalogen ***sound***.

  Exempel:
  ```
  Sound aSound = Sound.getSound("sound/race.mp3");
  if(aSound!=null) {
      aSound.play();
  }
  ```
  Om ljudfilen *sound/race.mp3* finns så returneras ett Sound-objekt.
  Om ljudfilen inte finns / inte finns stöd för kommer *sound* ha värdet *null*.

  Ljuduppspelningen startas med:
  `aSound.play();`

  Ljuduppspelningen stoppas med (kan ej startas på nytt):
  `aSound.stop();`

  Ljuduppspelningen kan pausas med `aSound.pause()` för att åter startas med `aSound.play()`.

  En ljudfil kan endast spelas en gång. När ljudfilen är slut så måste ett nytt Sound-objekt skapas (genom anrop till getSound-metoden).