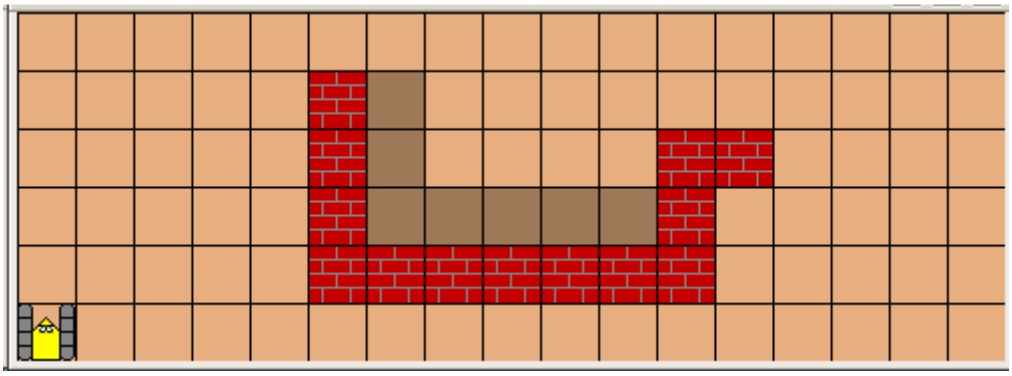
Laboration 2 TDA143

Syfte

Syftet med denna laboration är att öva på att använda metoder från en färdig klass för att lösa programmeringsproblem, att öva på att gradvis förfina ett program, och att bryta ner ett problem i enklare delproblem.

Inledning

I den här uppgiften ska ni programmera en simulerad robot att lösa ett par enkla uppgifter. Ni får tillgång till en handfull klasser för att köra simuleringen och visa en grafisk representaion av roboten, och ett antal textfiler som innehåller representationer av olika "världar" (klassen RobotWorld) som roboten kan röra sig i. Världarna är rektangulära nät av rutor (klassen Cell) som kan vara endera ljusa, mörka eller väggar (roboten kan inte passera genom väggar). Ni får gärna studera klasserna och försöka förstå hur de fungerar, men det är inte nödvändigt för att lösa uppgiften. Vid körning kan simuleringen se ut ungefär så här:



Roboten representeras i Java av ett objekt av typen Robot. Klassen har följande metoder:

void move()	Förflyttar roboten ett steg framåt. Om roboten hamnar utanför världen eller i en mur uppstår ett exekveringsfel (programmet krashar).
void turnLeft()	Vrider roboten 90° åt vänster.
void makeDark()	Färgar rutan roboten står på mörk. Kraschar om rutan redan är mörk.
void makeLight()	Färgar rutan roboten står på ljus. Kraschar om rutan redan är ljus.
boolean onDark()	Returnerar true om roboten står på en mörk ruta, annars false.
boolean canMove()	Returnerar true om det är möjligt för roboten att göra move() utan att ett exekveringsfel erhålls, returnerar annars false.
boolean atEndOfWorld()	Returnerar true om en förflyttning skulle innebära att roboten hamnade utanför världen.
Location getLocation()	Returnerar robotens position som ett objekt av typen Location.
int getDirection()	Returnerar robotens riktning (möjliga värden är Robot.EAST/WEST/SOUTH/NORTH).
void setDelay(int t)	Sätter hastigheten med vilken roboten rör sig. Parametern t anger antalet millisekunder varje rörelse tar att utföra.

Ni skall försöka och minimera användningen av upprepade eller långa instruktionssekvenser, och i stället deklarera lämpliga metoder (abstraktioner). Som en riktlinje bör de metoder ni definierar inte innehålla mer än 10 satser.

Metoderna ni utvecklar skall ha beskrivande namn och en kommentar som anger för- och eftervillkor, på det här viset:

```
//before: None
//after: Robot is facing opposite direction
public void turnAround() {
    robot.turnLeft()();
    robot.turnLeft();
}
```

Redovisning

Varje deluppgift ska redovisas för kursassistent på labbpass, för varje redovisning får ni en sjusiffrig kod. Därefter ska ni skicka in alla .java filer ni har skrivit eller ändrat, och fembokstäverskoderna. Följ instruktionerna på inlämningssidan på kurshemsidan. Sammanlagt ska ni skicka in tre fembokstäverskoder för Laboration 2.

Uppgift 1

I filen Cleaner.java finns ett program i vilket en värld och en robot skapas. Koden är som följer:

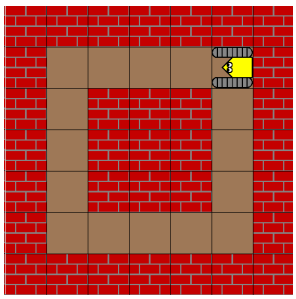
```
public class Cleaner {
   private Robot robot;
   public static void main(String[] args) {
      Cleaner cleaner = new Cleaner();
      cleaner.createEnviroment();
      cleaner.cleanCorridors();
   }

   private void createEnviroment() {
      RobotWorld world = RobotWorld.load("square.txt");
      robot = new Robot(new Location(1, world.getNumCols() - 2),
                        Robot.WEST, world);
      robot.setDelay(250);
   }


   //before: The room has four corridors, forming a square
   //        The robot is located in beginning of a corridor,
   //          facing the corridor in counter-clockwise direction.
   //        Each corridor has a length of five cells.
   //        All cells in the corridors are dark.
   //after:  All cells in the corridors are light.
   //        The robot has the same location as before and is facing
   //          the same direction
   private void cleanCorridors() {
      // This is where your code goes (remove this comment)
   }

}
```

Om du kompilerar och kör klassen visas ett fönster som ser ut så här:



Intenting annat händer, eftersom metoden cleanCorridors inte gör något ännu. Din uppgift är att fullborda denna metod utifrån specifikationen i kommentarerna. Därefter ska du i flera steg göra metoden mer användbar.

Klassen Cleaner innehåller en instansvariabel (robot) av klassen Robot, de båda instansmetoderna createEnvironment() och cleanCorridors(), samt en main-metod.

Metoden createEnviroment skapar först en värld. Världen är ett objekt av klassen RobotWorld som refereras via den lokala variabeln world. Världen skapas genom att anropa en statisk metod RobotWorld.load, som tar namnet på en textfil (square.txt) som parameter. Textfilen square.txt definierar hur världen kommer att se ut (läs gärna filen och försök förstå formatet). Sedan skapar metoden createEnviroment roboten som refereras via instansvariabeln robot:

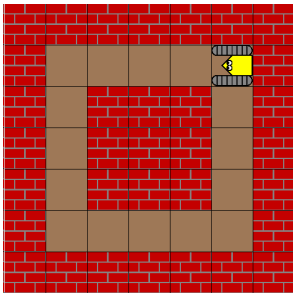
```
robot = new Robot(1, world.getNumCols()-2, Robot.WEST,world);
```

Den första parametern till konstruktorn (`new Location(1, world.getNumCols() - 2)`) anger i vilken position i världen roboten skall placeras. Den andra parametern (Robot.WEST) anger robotens initiala riktning och den sista parametern (world) anger vilken värld roboten skall bebo.

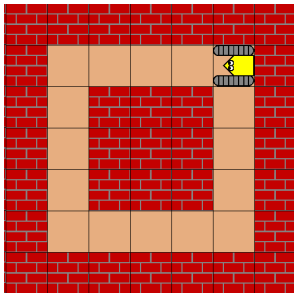
I världen har cellen längst upp till vänster positionen (0,0). Det är möjligt att fråga världen hur stor den är. Genom att anropa metoden getNumCol() tar vi reda på hur många celler världen har i horisontell led (= antalet kolumner). Om vi vill ta reda på antalet celler världen har i vertikal led (=antal rader) finns metoden getNumRow(). Koordinaterna i det här anropet anger att roboten är placerad på rad 1 (näst längst norrut) på kolumnen näst längst österut.

Slutligen gör metoden createEnviroment ett anrop av metoden setDelay för att ange med vilken hastighet roboten skall röra sig. Parametern till setDelay anger hur många millisekunder det tar att utföra en rörelse (sväng, färgändring eller förflyttning). Så lägre värde ger snabbare robot.

1 a) Din uppgift är att komplettera metoden cleanCorridors(), på så sätt att roboten genomlöper korridorerna moturs och ”städar” dessa genom att sätta de mörka cellerna till ljusa (enligt figurerna nedan).



Före



Efter

Du får anta att samtliga korridorer är 5 celler långa och att alla celler är mörka. Vidare får du anta att roboten står i ett hörn och är riktad på så sätt att den har ytterväggen på sin högra sida (vilket innebär att en förflyttning av roboten sker i moturs riktning i korridoren). Samtliga dessa antaganden är förvillkor till metoden cleanCorridors(). När roboten utfört sin uppgift skall den befinna sig i samma tillstånd (dvs ha samma position och riktning) som den hade innan uppgiften utfördes. Din lösning på den här deluppgiften får endast använda instruktionerna move(), turnLeft() och makeLight() till roboten.

1 b) Spara undan en kopia av lösningen från deluppgift a) i filen CleanerA.java.

Förändra ditt program (Cleaner.java) från deluppgift a) på så sätt att programmet klarar av att handha korridorer som redan delvis är ”städade”, dvs där några celler initialt är ljus. Förvillkoret "all cells in the corridors are dark" till metoden cleanCorridors() skall alltså tas bort.

Glöm inte att uppdatera förvillkoren. Programmet skall testas med världen som finns beskriven i filen square2.txt. Du behöver därför också byta ut filnamnet square.txt mot square2.txt i metoden createEnviroment().

Du får använda samtliga instruktioner roboten erbjuder nu.

1 c) Spara undan en kopia av lösningen från deluppgift b) i filen CleanerB.java.

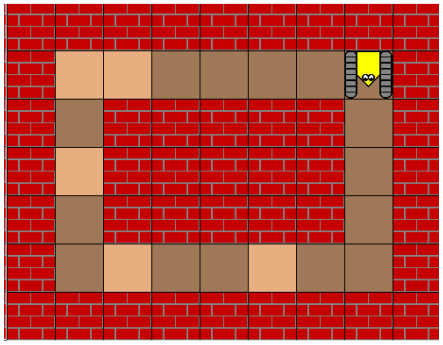
Nu skall du förändra ditt program från deluppgift b) på så sätt att roboten klarar att ”städa” byggnader där de horisontella och vertikala korridorerna är olika långa (se figuren nedan).

Förvillkoret ”each corridor has a length of five cells” till metoden cleanCorridors() tas alltså bort.

Du skall testa ditt program med världen som beskrivs av filen square3.txt.

1 d) Spara undan en kopia av lösningen från deluppgift c) i filen CleanerC.java.

Nu skall du förändra ditt program från deluppgift c) på så sätt att förvillkoret ”facing the corridor in counter-clockwise direction” till metoden CleanCorridors() tas bort. Roboten skall alltså klara av att ”städa” korridorerna även medurs (se figuren nedan).



Du skall testa ditt program med världen som beskrivs av filen square3.txt. Testa ditt program för såväl då robotens rörelseriktning är moturs som medurs. För att få roboten att gå medurs ändras robotens startriktning till Robot.SOUTH.

1 e) Spara undan en kopia av lösningen från deluppgift d) i filen CleanerD.java.

Nu skall du förändra ditt program från deluppgift d) på så sätt att det klarar av att roboten placeras på en godtycklig position i någon korridors längdriktning (se figuren bredvid).

Förvillkoret ”the robot is located in beginning of one of the corridors” till metoden cleanCorridors() tas alltså bort.

Testa programmet med samma värld och med samma initiala placering av roboten som i deluppgift d) . Testa också programmet för olika startriktningar, samt roboten placerad mitt på en korridor (det du behöver göra är att ändra parametern world.getNumCols() - 2 i metoden createEnviroment() till exempelvis värdet 4).

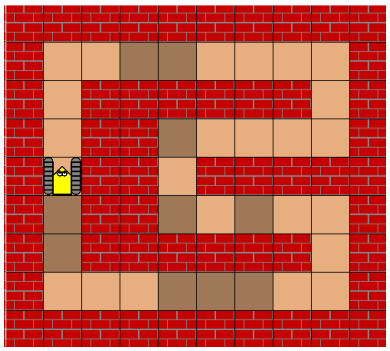
Tips: För att uppfylla eftervillkoret ”the robot has the same location” till metoden cleanCorridors() är det nödvändigt att spara undan robotens startposition. Detta görs med hjälp av operationen getLocation(), som returnerar positionen som ett objekt av klassen Location. För att kontrollera om

två objekt startPosition och currentPosition av klassen Location är lika används instansmetoden equals enligt **`startPosition.equals(currentPosition)`**. Notera: att ha en variabel som heter currentPosition är inte nödvändigt, eller en särskilt bra idé! Fundera på vilket uttryck man borde använda istället. Även robotens initiala riktning kan behöva sparas undan och återställas i slutet av programmet.

1 f) Spara undan en kopia av lösningen från deluppgift e) i filen CleanerE.java.

Slutligen skall vi ta bort förvillkoret “the room has four corridors, forming a square” till metoden cleanCorridors() och ersätta detta med förvillkoret “the corridors form a closed loop”.

Roboten skall alltså klara av att ”städa” korridorer enligt scenariot i figuren nedan:



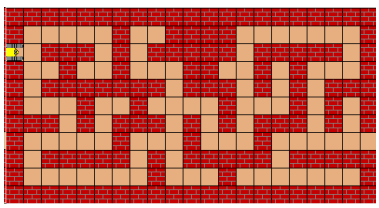
Testa programmet med världen som beskrivs av filen loop.txt, med lite olika startpositioner och riktningar.

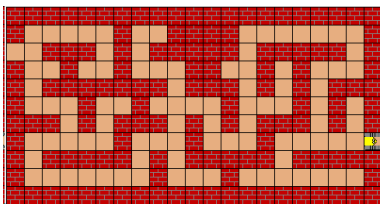
Tips: Det finns ett par olika lösningar på problemet. En är att göra en metod som alltid flyttar ett steg framåt i en tunnel, det vill säga en metod som antar att det finns två möjliga riktningar att förflytta sig, och den flyttar sig endera framåt, vänster eller höger, men aldrig till en som är bakåt. Sedan är det bara att upprepa den tills roboten är tillbaks där den började.

1 g) Städa i ordning er kod innan ni redovisar. Har ni följt instruktionerna korrekt har ni kopior CleanerA/B/C/D/E.java av koden, samt Cleaner.java som innehåller körbar lösning på f). Är koden onödigt krånglig? Förenkla den. Se särskilt till att ni följer riktlinjen om hur lång varje metod får vara (runt 10 satser, räkna semikolon). Se till att ni har vettiga variabelnamn överallt och att ni indenterat korrekt.

Uppgift 2

I filen MazeFinder.java finns ett program i vilket en värld och en robot har skapats. Roboten befinner sig i ingången till en labyrint och uppdraget roboten skall utföra är att förflytta sig till utgången av labyrinten (enligt figurerna nedan).



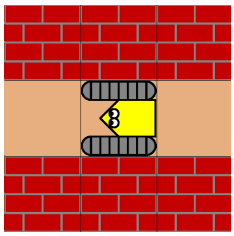


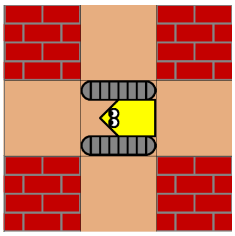
Före Efter

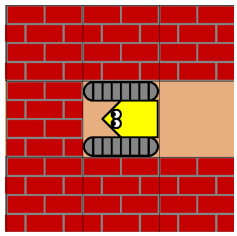
Din uppgift är att komplettera metoden findExit().

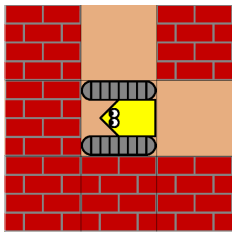
Problemet med att ta sig igenom en labyrint, som är enkelt sammanhängande (dvs där vägarna inte bildar några cykler) kan lösas genom att följa ena väggen. Man skall alltså se till att alltid ha en vägg på på sin vänstra sida (eller att alltid ha en vägg på sin högra sida).

För att kunna följa en vägg (i detta fall på den vänstra sidan), beakta vilka olika fall det finns och hur roboten ska bete sig, till exempel i de här olika fallen:


Flytta ett steg framåt


Sväng vänster och flytta


Vänd om och flytta


Sväng höger och flytta

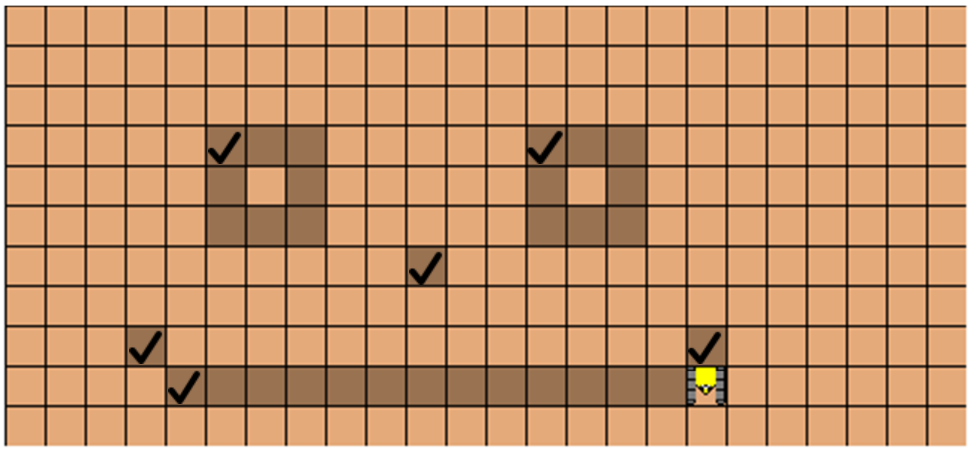
Tips: Snabba på animeringen lite (minska värdet till setDelay() om roboten är för långsam.

Uppgift 3

I den här uppgiften ska ni få er robot att rita en vacker bild. Mer allmänt ska ni skriva ett antal metoder för att programmera roboten att rita bilder. Klassen Drawing laddar canvas.txt, en helt tom yta.

Ni har stor frihet att bestämma vilka metoder ni definierar, men ett minimikrav är en metod som ritar en kvadrat med sitt nordvästra hörn i en bestämt cell (en Location) och med en bestämd längd på sidorna. Metoden ska i sin tur anropa andra enklare ritmetoder.

För att visa att det fungerar ska ni göra så att metoden draw() ritar följande figur:



För att hjälpa er på traven lite så kan jag visa hur metoden draw såg ut i min implementation:

```
public void draw() {
    drawSquare(new Location(3,5), 3);
    drawSquare(new Location(3,13), 3);
    drawSquare(new Location(6,10), 1); // Nose
    drawSquare(new Location(8,3), 1);
    moveTo(new Location(9,4));
    turn(Robot.EAST);
    drawLine(13);
    drawSquare(new Location(8,17),1);
    robot.move();
}
```

Programmet ska aldrig krascha av att man ritar över en cell som redan är mörk, men om man flyttar roboten utanför området kan programmet krascha.

Tips: Om ni redan har en metod för att rita ett rakt sträck, en metod för att flytta roboten till en bestämd plats, och en metod för att rotera roboten till en bestämd riktning så är det enkelt att implementera **`drawSquare`**.

Bonusuppgift:

Den här uppgiften är helt frivillig, och behöver inte redovisas annat än för att imponera på Jonas eller någon av kursassistenterna.

I Hunt.java finns ett litet program som förklarar uppgiften. Fungerar det som det ska bör det visa en robot som står still och en liten irriterande grej som far omkring av sig själv. Din uppgift är att programmera roboten att så snabbt som möjligt