Lab 5

Stack

Agenda

- Line following demo
  - Make sure you take a video
  - Submit your video and code on GitHub
- Team update: this is the week you can vote out a team member
- Intro to stack and switch
- Project introduction

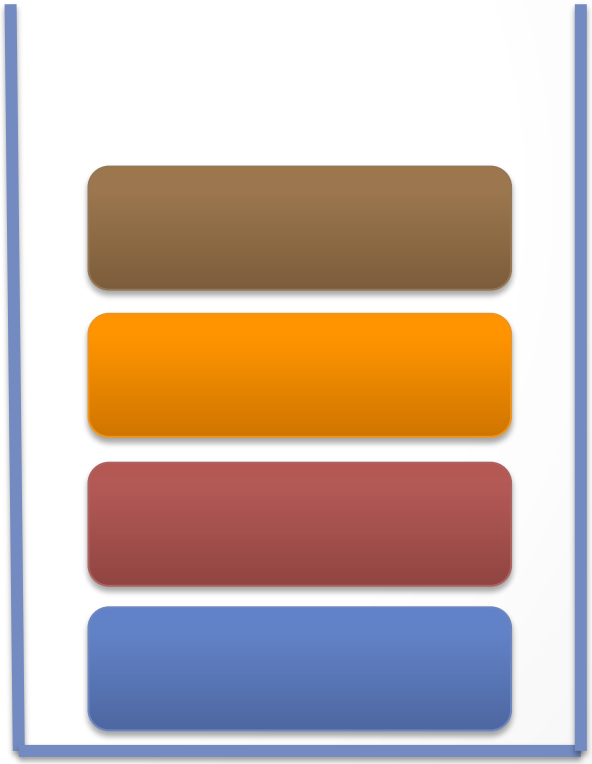
Stack

- Used in manipulating arbitrarily large collections of objects.
- A *stack* is a collection that is based on the last-in-first-out (LIFO) policy.
- We name the stack *insert* method `push()`, and the stack *remove* operation `pop()`
- We also have a method to test whether the stack is empty.

STACK API

public class Stack<Item> implements Iterable<Item>	
Stack()	*create an empty stack*
boolean isEmpty()	*is the stack empty?*
void push(Item item)	*push an item onto the stack*
Item pop()	*return and remove the item that was inserted most recently*
int size()	*number of items on stack*

How it works



How to use?

- Import

```
import java.util.Stack;
```

- Declaration:

```
Stack<Object> var_name = new Stack<Object>();
```

  - If we want a stack of int:

```
Stack<Integer> path = new Stack<Integer>();
```

- Usage

```
path.push(0); //put value 0 on stack
int top_value = path.peek(); //get the value at the top of the stack
path.pop(); //remove top value from stack
```

Example

```
import java.util.Stack;

public class StackDemo {

     public static void main(String[] args) {
          //Declare a stack that holds int data type
          Stack<Integer> path = new Stack<Integer>();
          System.out.println("Empty stack : "  + path.isEmpty());
          path.push(1); //put 1 on the stack
          System.out.println("Stack : "  + path);
          path.push(2); //put 2 on the stack
          System.out.println("Stack : "  + path);
          path.push(3); //put 3 on the stack
          System.out.println("Stack : "  + path);
          int top_value = path.peek(); // should be 3
          System.out.println("At the top of the stack : "  + top_value);
          while(!path.isEmpty()){
              path.pop(); //take the top element out
              System.out.println("Stack after pop: "  + path);
          }
     }
}
```

```
Empty stack : true
Stack : [1]
Stack : [1, 2]
Stack : [1, 2, 3]
At the top of the stack : 3
Stack after pop: [1, 2]
Stack after pop: [1]
Stack after pop: []
```

Switch Statement

```
switch(expression)
{
        case value1:{
                // statements
                break;
        }
        case value2:{
                // statements
                break;
        }
        default:{
                // statements
                break;
        }

}
```

Stack and Switch Project

- Let's assume that for each value we pushed on the stack there is an equivalent action we want to do:
  - 1: move forward 10
  - 2: turn left
  - 3: turn right
- How to execute those commands?

```
while(!path.isEmpty()){
    top_value = path.peek();
    path.pop();
    switch(top_value)
    {
        case 1:{   //if at the top of the stack the value is 1 move 10
                pilot.travel(10);
                break;
        }
        case 2:{  //if at the top of the stack the value is 2 rotate 90
                pilot.rotate(90);
                break;
        }
        case 3:{ //if at the top of the stack the value is 3 rotate -90
                pilot.rotate(-90);
                break;
        }
        default:{  //default action: stop
                pilot.stop();
                break;
        }
    }
}
```

Project Description

- The problem to be solved is a maze navigation problem
- You will be given a maze with a starting cell and a goal cell.
- Your job is to come up with a strategy and implement the corresponding algorithm so that your robot will go from the starting cell to the goal cell, acknowledge that it reached the goal cell with some sort of action, and then make its way back to the starting cell without making a wrong decision.

Maze Specification

- The maze can be described as a 5 x 4 grid of cells where each cell is 11.5 inches x 11.5 inches.
- There are 3 types of cells:
  - The floor cells contain a black line going through the center of the cell leading to the next cell.
  - The starting cell will be empty.
  - The goal cell is covered with aluminum foil.
  - The intersections are marked with a red color tape.
- Your robot will begin in the starting cell.
- Once your robot has reached the goal cell, it should acknowledge that it has reached the endpoint by performing some action (beeping, making a specific move, print something to the screen, etc.). Then the timer will start to record how long it takes to return to the start cell.
- There is no limit to the number and place of T-junctions and corners. Also, there may be dead-ends.