Data Structure

Fall 2019

M 16:00-18:00 W 11:00-13:00

http://smart.hallym.ac.kr

Instructor: **Jin Kim**
010-6267-8189(033-248-2318)
jinkim@hallym.ac.kr

Office Hours:

Lab(Stack2)

Fall 2019

http://smart.hallym.ac.kr

Instructor: **Jin Kim**
010-6267-8189(033-248-2318)
jinkim@hallym.ac.kr

Office Hours:

Java Stack

- https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html
- 여러 static (정적) 메소드 가 존재한다.

Java.util.Stack

```
import java.util.Stack;
….
 Stack<Integer> stack1 = new Stack<>();

….
stack1.push( c);
….
stack1.pop()
```

java.util.Stack 사용하기

- 자바에서는 java.util.Stack에서 스택관련 메소드를 아래와 같이 제공한다.
  - Void push(Object item)
  - Object pop()
  - Object peek()
  - Int size()
  - Boolean empty()
- 다음과 같은 테스트를 해보라
  - Push(10), push(20), push(30), pop(), peek(), push(40), push(pop()) 등

java.util.Stack 사용하기

Generic : 클래스 내부에서 사용할 데이터 타입을
나중에 인스턴스를 생성할 때 확정하는 것을 제네릭이라 한다

정수 사용

```
import java.util.Stack;
…
Stack<Integer> s = new Stack<>();
```

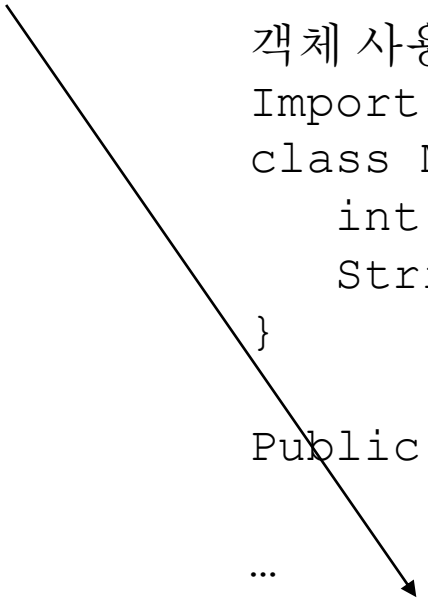
컴파일시에 확정

문자열 사용

```
import java.util.Stack;
…
Stack<String> s = new Stack<>();
```

java.util.Stack 사용하기

Generic : 클래스 내부에서 사용할 데이터 타입을
나중에 인스턴스를 생성할 때 확정하는 것을 제네릭이라 한다



객체 사용

```
Import java.util.Stack;
class MyNode {
    int i;
    String name;
}

Public class MyStack {

…
Stack<MyNode> s = new Stack<>();

MyNode node1 = new MyNode();
s.push(node1)
MyNode node2 = s.pop();
```

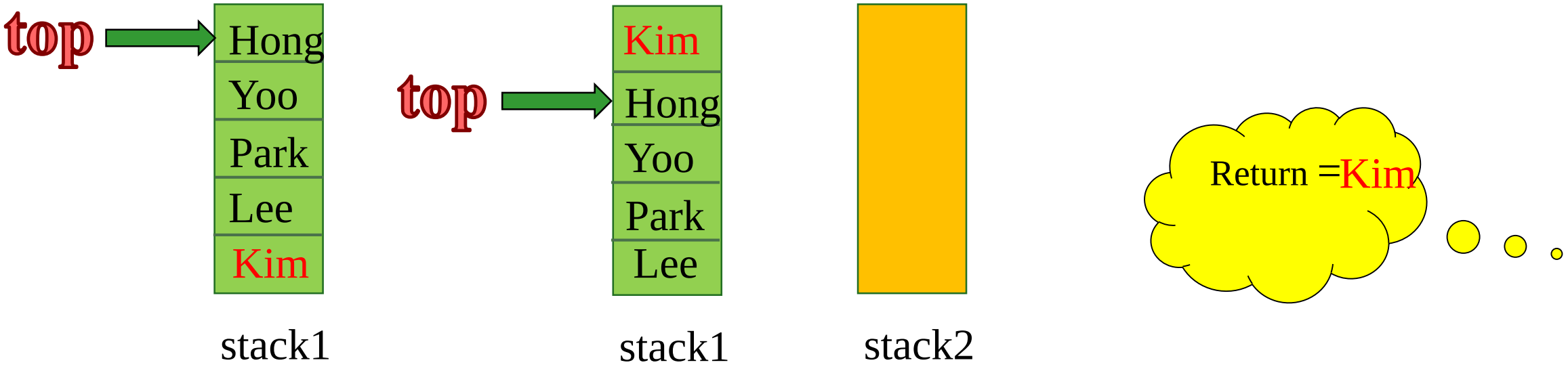
(1, “Kim”), (2, “Lee”), (3, “Choi)를 push한 후, pop하고 출력하라. MyGeneric.java

자바에서 제공하는 java.util.Stack 사용

- 중위표기식을 후위표기식으로 바꾸어보자. InfixToPostfix.java와 InfixToPostfixMain.java를 사용하라. 다양한 중위표기식을 만들어 테스트해보라.
- 후위연산자 계산을 UtilStack.java와 PosExpMain.java를 이용해보자
- 두 프로그램의 main 부분을 수정하여 위의 과정을 연속하여 계산하도록 하는 Main 프로그램을 만들어라. 그 프로그램의 이름을 InfixMain.java라 하자
- 1*(3-2)+((6-2)/4)를 계산하라. 중위연산식을 후위연산식으로 변경하고, 변경된 후위식을 계산하라.
- 위의 두 프로그램을 한 프로그램으로 합쳐 보라(MyInfixMain.java).

두 개의 스택으로 큐 시뮬레이션

- 스택 s1, s2를 만들어라. 스택 s1에 원소를 여러 개 push한다. 우리는 스택의 제일 밑부분원소를 꺼내어 선두로 보내고 싶다. 스택 두 개를 사용하여 이를 해결하라. 이 프로그램을 MyQueueSim.java라 하라. 스택 s2는 임시 스택
- 다음과 같은 연산이 유용할 것 같다. s2.push(s1.pop()), s1.push(s2.pop())



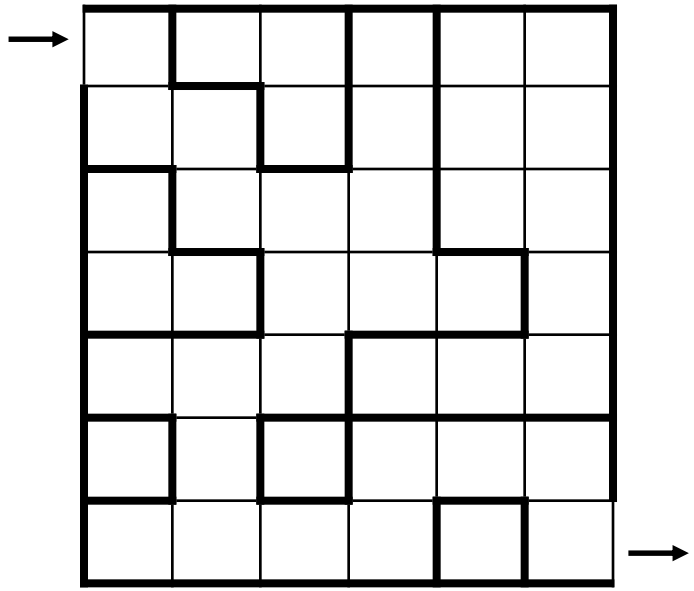
미로 문제 (1)

- Maze(미로)
  - 예

입구

0	x	x	0	x	x
0	0	x	0	x	x
x	0	0	0	x	x
x	x	0	0	0	x
0	0	0	x	x	x
x	0	x	0	0	0
0	0	0	0	x	0

출구



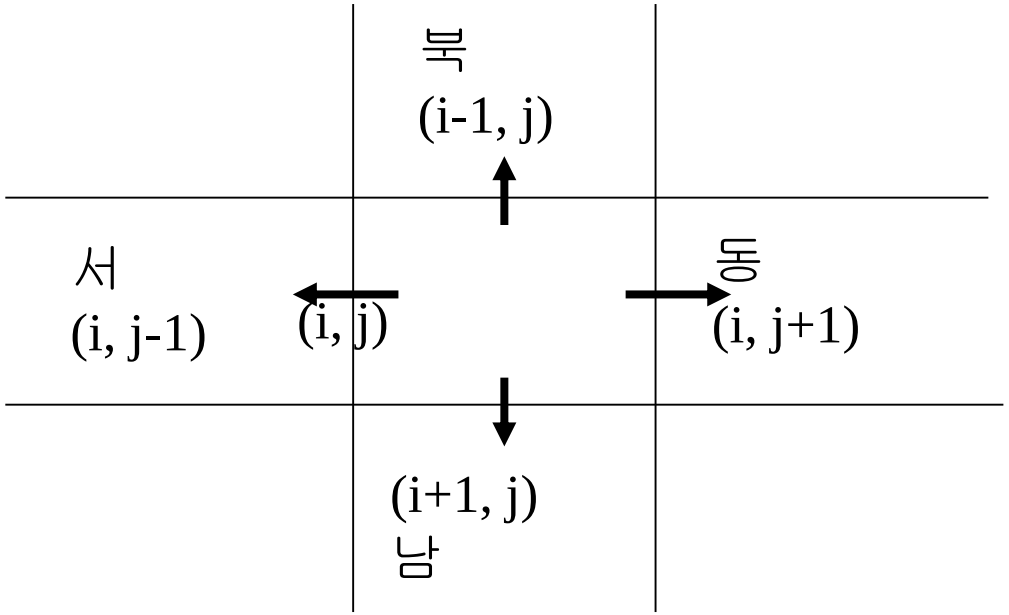
m X n 미로를 maze(m+2, n+2) 배열로 표현

- 사방을 1로 둘러싸서 경계 위치에 있을 때의 예외성(두 방향만 존재)을 제거

Maze 미로 문제 (2)

- Current position현재 위치 : maze[i][j]

- Moving direction(이동 방향)
  - North,East,South,West(북, 동, 남, 서 순서) (clockwise시계 방향)



Maze 미로 문제 (3)

- Move direction array(이동 방향 배열) : move[4, 2]

(dir)	i[0]	j[1]
N북[0]	-1	0
E동[1]	0	1
S남[2]	1	0
W서[3]	0	-1

- Next position calculation(다음 위치계산) : maze[nexti,nextj]
  - next_i ← i + move[dir, i]
  - next_j ← j + move[dir, j]
- Visited position saving to (방문한 경로를) mark[m+2, n+2]에 저장
  - 한 번 시도했던 위치로는 다시 이동하지 않음
- Visited path stack save 지나온 경로의 기억 <i, j, dir>을 스택에 저장
  - 스택의 최대 크기 : m * n

미로 경로 발견 알고리즘 (1)

```
mazePath( )
   maze[m+2, n+2];// m × n 크기의 미로 표현
                  // 행 0과 m+1, 열 0과 n+1은 1로 초기화
                  // 입구는 maze[1, 1], 출구는 maze[m, n]
   mark[m+2, n+2]; // 방문 위치를 표시할 배열로 모든 원소를 0으로 초기화
      // dir = {0(북), 1(동), 2(남), 3(서)}
   stack[m×n];
   top ← -1;    // 3원소 쌍 <i, j, dir>을 저장하는 stack을 초기화
   push(stack, <1, 1, 1>); //입구 위치 (1,1), 이동 방향은 동(1)으로 초기화
  mark[1,1]=1;
   while (not isEmpty(stack)) do {        // 스택의 공백 여부를 검사
      <i, j, dir> ← pop(stack);           // 스택의 톱 원소를 제거
      while (dir ≤ 3) do {        // 시도해 볼 방향이 있는 한 계속 시도
         next_i ← i + move[dir, i];       // 다음 시도할 행(i)을 설정
         next_j ← j + move[dir, j];       // 다음 시도할 열(j)을 설정
         if (next_i = m and next_j = n)    // 미로 경로 발견
         then {
              print("The path is as follows.");
              print(path in stack);  // 스택에 저장된 경로 출력
              print(i, j);   print(m, n);
            return;
         }
```

미로 경로 발견 알고리즘 (2)

```
            if (maze[next_i, next_j] = 0 and        // 이동 가능 여부 검사
                mark[next_i, next_j] = 0 // 시도해 보지 않은 위치인지 검사
               then {
                mark[next_i, next_j] ← 1;
                push(stack, <i, j, dir>);              // 이동한 위치를 스택에 기록
                <i, j, dir> ← <next_i, next_j, 0>;
               }
               else dir ← dir + 1;   // 다음 이동할 방향을 설정
        }
    }
    print("There is no path");  //미로 경로가 없는 경우

end mazePath()
```

교과서에 있는 미로 프로그램

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

class MazeCell{
 int i;
 int j;
 int dir;
 public MazeCell(int  _i,int  _j,int _dir){
  i = _i;
  j = _j;
  dir = _dir;
 }
 //public String toString(){
 // return "<" + i  + "," + j + ","+dir+">";
 //}
}
```

MazePath.java

```
public class MazePath
{
 private int[][] maze = {
       {1,1,1,1,1,1,1,1,1,1},//1
       {1,0,1,1,0,1,1,1,1,1},//2
       {1,0,0,1,0,1,1,1,1,1},//3
       {1,1,0,0,0,1,1,1,1,1},//4
       {1,1,1,0,0,0,1,1,1,1},//5
       {1,0,0,0,1,1,1,1,1,1},//6
       {1,1,0,1,0,0,0,1,1,1},//7
       {1,0,0,0,0,1,0,0,1},//8
       {1,1,1,1,1,1,1,1,1,1}  //9
       };
 private int[][] move ={{-1,0},{0,1},{1,0},{0,-1}};//북 동 남 서
 private int m = maze.length -2;
 private int n = maze[0].length -2;
```

```
public MazePath(){  //알고리즘대로 프로그램 작성
  int mark[][] = new int[maze.length][maze[0].length];
  Stack st = new Stack();
  //Stack st1 = new Stack();  //올바른 경로 삽입할 스택
  st.push(new MazeCell(1,1,1));  //초기출발위치및 방향 설정
  while(st.isEmpty()!=true){
   MazeCell  mc = (MazeCell)st.pop();  //지나온 경로를 pop함
   while(mc.dir<=3){
    int nextI = mc.i + move[mc.dir][0];  //다음 시도할 행(i)를 설정
    int nextJ = mc.j + move[mc.dir][1];  //다음 시도할 열(j)를 설정

    if(nextI == m && nextJ ==n){  //미로경로 발견
     System.out.println("The path is as follows");
     st.push(new MazeCell(mc.i,mc.j,mc.dir)); //마지막 경로 지정
     st.push(new MazeCell(nextI,nextJ,0));    //마지막 지점 지정
      while(st.isEmpty()!=true){
                    //st1.push((MazeCell)st.pop());
                    mc=(MazeCell)st.pop();
                    maze[mc.i][mc.j]=9;
                    System.out.println(mc);
      }
     //while(st1.isEmpty()!=true){
     // mc = (MazeCell)st1.pop(); //올바른 경로 출력
     // System.out.println(mc);
      //maze[mc.i][mc.j]=9;
     //}
     displayMaze();  //경로 확인 완료후 출력
     return;
    }
    if(maze[nextI][nextJ]==0 &&
mark[nextI][nextJ]==0){//이동가능&시도해보지 않은위치
      mark[nextI][nextJ] = 1;
      st.push(new MazeCell(mc.i,mc.j,mc.dir));  //지나온경로를 스택에 push
      //mc = new MazeCell(nextI,nextJ,0);  //새로운 위치와 방향 설정 방향은
초기화
      mc.i = nextI;
      mc.j = nextJ;
      mc.dir = 0;
     }
     else
     {
      mc.dir++;
     }
    }
   }
   System.out.println("There is no path");

  }
```

```
public void displayMaze(){
  String[] str = {"x","x","9"};
  for(int i = 0 ; i < maze.length ; i++){
   for(int j = 0 ; j < maze[0].length ; j++){
                        //System.out.print(str[(maze[i][j])]);
                        System.out.print((maze[i][j]));
   }
   System.out.println("");
  }
 }

 public static void main(String[] args)
 {
  new MazePath();
 }
}
```

자바의 모든 객체, 클래스는 자동적으로 Object로부터 상속받음.
이 Object에는 모든 다른 객체들에게 공통적으로 필요한 메서드들이 정의되어있으면서,
다른 모든 자바 클래스에서는 이 메서드들을 오버라이딩하여 사용하도록 약속되어 있다.
그 중에 String toString() 메서드가 있다. 이 메서드는 객체를 문자열 형태로 표현한 String을 반환함.

- 다른 미로를 만들고 정상 작동이 되는지 확인하라

너무도 아름다운,감탄이 절로나오는 미로프로그램(재귀)

MyMaze.java

아래 프로그램에 이차원으로 된 방문경로를 출력하는 코드를 추가하라.

```
public class maze {
        static int reached=0;
        static int[][] check= new int[7][7];
        static int maze[][]={
                        {1,1,1,1,1,1,1},
                        {1,0,1,0,0,0,1},
                        {1,0,1,0,1,0,1},
                        {1,0,0,0,1,0,1},
                        {1,0,1,1,1,0,1},
                        {1,0,0,0,1,0,1},
                        {1,1,1,1,1,1,1}
                        };
        static void back(int x, int y){
                if( reached == 1 ) return;      //목적지에 도착했으면 처리 없이 return
         System.out.printf("(%d,%d)\n",x,y);     // stack의 이동상황을 출력해주는 출력문
           if( x == 5 && y == 5 ){
              reached = 1;
              return;
           }
           check[x][y] = 1;        //현재 위치 방문 했기때문에, check
           if( maze[x+1][y] == 0 && check[x+1][y] == 0 ) back(x+1,y); // 아래로
           if( maze[x-1][y] == 0 && check[x-1][y] == 0 ) back(x-1,y); // 위로
           if( maze[x][y+1] == 0 && check[x][y+1] == 0 ) back(x,y+1); // 오른쪽으로
           if( maze[x][y-1] == 0 && check[x][y-1] == 0 ) back(x,y-1); // 왼쪽으로
           check[x][y] = 0;

        }
        public static void main(String[] args){
                back(1,1);
        }
}
```

```
{1,1,1,1,1,1,1},
{1,2,1,0,0,0,1},
{1,2,1,0,1,0,1},
{1,0,0,0,1,0,1},
{1,0,1,1,1,0,1},
{1,0,0,0,1,0,1},
{1,1,1,1,1,1,1}
};
```

$a^n b^n$

- 문자열 $a^n b^n$을 인식하는 프로그램을 작성하라. (ab, aabb, aaabbb, ….) 참이면 true, 거짓이면 false
- 문자열 $a^n b^{2n} c^n$ 을 인식하는 프로그램을 작성하라. (abbc, aabbbbcc, aaabbbbbbcccc, ….) (MyStringCheck.java)

1. MyGeneric.java
2. MyQueueSim.java
3. MyMaze.java
4. MyStringCheck.java

Upload your program