**JAVA 실습 14주차**

20234014 컴퓨터 공학과 이은정

1. ArrayList 페어

import java.util.ArrayList;

class SimplePair <T>{

    ArrayList<T> list;

    SimplePair(T text, T text2){

        super();

        list = new ArrayList<>();

        list.add(text);

        list.add(text2);

    }

    public T getFirst() {

        return list.get(0);

    }

    public T getSecond() {

        return list.get(1);

    }

}

public class SimplePairTest {

    public static void main(String[] args) {

        SimplePair<String> pair = new SimplePair<String>("apple", "tomato");

        System.out.println(pair.getFirst());

        System.out.println(pair.getSecond());

    }

}



위 코드는 SimplePair라는 클래스를 통해 2개의 String 객체를 저장할 수 있다. 이 때 simplePair는 제네릭 변수를 사용해 무슨 변수형이든 저장할 수 있다. 이 때 변수는 String, Boolean이라고 쓰지 않고 T를 사용해 설정한다. ArrayList 처럼 무슨 변수형을 넣어도 작동할 수 있어야 하므로 SimplePair<T>라고 변수 이름을 설정한다. 메서드 getFirst, getSecond를 통해 값을 반환할 때에도 T 형식을 사용해야 한다.

2. 합, 교집합

import java.util.Collection;

import java.util.HashSet;

import java.util.Iterator;

import java.util.Set;

public class main {

    public static void main(String[] args) {

        Set<String> s1 = new HashSet<String>();

        s1.add("A");

        s1.add("B");

        s1.add("C");

        Set<String> s2 = new HashSet<String>();

        s2.add("A");

        s2.add("D");

        Set<String> s3 = new HashSet<String>();

        for (String s: s1)

            s3.add(s);

;       s3.addAll(s2);

        System.out.println("합집합: "+s3);

        System.out.println("\n");

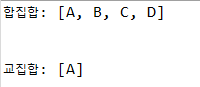
        s1.retainAll(s2);

        System.out.println("교집합: "+s1);

        System.out.println("\n");

    }

}



Set 클래스에서는 합, 교집합 메서드를 사용해 원본 데이터를 변경할 수 있다. 따라서 합집합을 계산한 이후, 교집합을 계산하면 이미 변형된 데이터로 교집합을 계산하게 되므로, s1, s2에 값을 저장한 이후에 s3에 s1를 데이터를 저장한다, 이 때 s3 = s1를 하게 되면 s1이 가리키는 주소를 s3도 같이 가리키게 되므로 s1의 원본을 변경하게 된다. 따라서 s3에 s1의 데이터를 하나하나 가져와 s1를 가져와 저장한다.

이후 s3에서 s2에 addAll을 통해 합집합을 계산하고, s1에 s2에 retainAll을 통해 교집합을 계산할 수 있다.

3. 국가 해시맵

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class Coutnry {

    public static void main(String[] args) {

        Map<String, String> countryCapitalMap = new HashMap<>();

        Scanner sc = new Scanner(System.in);

        countryCapitalMap.put("USA", "Washington");

        countryCapitalMap.put("Japan", "Tokyo");

        countryCapitalMap.put("China", "Beijing");

        countryCapitalMap.put("UK", "London");

        countryCapitalMap.put("Korea", "Seoul");

        for (int i = 0; i < 2; i++) {

            System.out.print("국가 이름을 입력하시오: ");

            String county = sc.next();

            if (countryCapitalMap.containsKey(county))

                System.out.println(county + "의 수도 " + countryCapitalMap.get(county)+"\n");

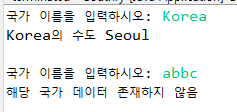
            else

                System.out.println("해당 국가 데이터 존재하지 않음\n");

        }

    }

}



countryCapitalMap에 String 형으로 key, value값을 넣어준다. Scanner 변수를 통해 내가 검색할 국가 변수를 String 변수에 저장한다. 이후에 countryCapitalMap에서 containsKey를 통해 해당 key를 검색한다. 만약 해당 값이 true 즉슨 값이 존재한다면 getKey를 통해 해당 value를 가져온다. 만약 존재하지 않는다면 데이터가 존재하지 않는다고 안내한다.

4. 리스트 시간 측정

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.List;

import java.util.Random;

public class ListPerformanceComparison {

    private static final int ELEMENTS\_COUNT = 100000;

    private static final int OPERATIONS\_COUNT = 10000;

    public static void main(String[] args) {

        List<Integer> arrayList = new ArrayList<>();

        List<Integer> linkedList = new LinkedList<>();

        Random random = new Random();

        // ArrayList add operation

        long startTime = System.nanoTime();

        for (int i=0; i<OPERATIONS\_COUNT;i++)

            arrayList.add(0, random.nextInt(ELEMENTS\_COUNT));

        long endTime = System.nanoTime();

        System.out.println("ArrayList add: " + (endTime - startTime) / 1e6 + " ms");

        startTime = System.nanoTime();

        for (int i=0; i<OPERATIONS\_COUNT;i++)

            linkedList.add(0, random.nextInt(ELEMENTS\_COUNT));

        endTime = System.nanoTime();

        System.out.println("linkedList add: " + (endTime - startTime) / 1e6 + " ms");

    }

}



위 코드는 각 리스트에 0~9999 개의 데이터 중 랜덤한 하나의 값을 10000번 저장한다. 위 결과를 보면 ArrayList가 시간이 더 오래 걸리는 것을 볼 수 있다. ArrayList는 for문을 사용해 n개의 1번 인덱스 이후 데이터를 모두 옮겨야 해서 O(n)의 시간 복잡도를 가지지만, link 를 통해 0번 인덱스에 데이터 저장 시 링크만 바꾸면 되기 때문에 O(1)의 시간 복잡도를 가진다. 따라서 link를 사용한 리스트에서 조금 더 빠르게 완료되는 것을 볼 수 있다.

5. 몬스터 캐치 게임

package week12\_04;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.util.ArrayList;

import java.util.Iterator;

import javax.swing.ImageIcon;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.Timer;

class Monster {

    int x, y, hp;

    Image img;

    int speed;

    Monster(int x, int y, int hp) {

        this.x = x;

        this.y = y;

        this.hp = hp;

        ImageIcon icon = new ImageIcon(getClass().getResource("/Image/monster.png"));

        img = icon.getImage();

        speed = (int)(Math.random()\*10+5);

        if (Math.random()<0.5)

            speed\*=-1;

    }

    public void draw(Graphics g) {

        g.drawImage(img, x, y, null);

    }

}

class MyFrame extends JFrame {

    static ArrayList<Monster> monsters;

    MyPanel panel;

    MyFrame() {

        monsters = new ArrayList<Monster>();

        this.setTitle("Monster Catch Game");

        this.setSize(700, 300);

        for (int i = 0; i < 10; i++) {

            monsters.add(new Monster((int) (Math.random() \* 600), (int) (Math.random() \* 200), 1));

        }

        panel = new MyPanel();

        this.add(panel);

        this.addMouseListener(new MouseAdapter() {

            @Override

            public void mouseClicked(MouseEvent e) {

                int x = e.getX();

                int y = e.getY();

                Iterator<Monster> iterator = monsters.iterator();

                while (iterator.hasNext()) {

                    Monster m = iterator.next();

                    if (x >= m.x && x <= m.x + m.img.getWidth(null) &&

                        y >= m.y && y <= m.y + m.img.getHeight(null)) {

                        iterator.remove();

                    }

                }

                repaint();

                if (monsters.size()==0)

                    System.exit(1);

            }

        });

        Timer timer = new Timer(100, e -> {

            move();

            repaint();

        });

        timer.start();

        this.setLocationRelativeTo(null);

        this.setVisible(true);

        this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

    }

    void move() {

        for (Monster m : monsters) {

            m.x += m.speed;

            m.y += m.speed;

            if (m.x < 0 || m.x > getWidth() - m.img.getWidth(null)) {

                m.speed \*= -1;

            }

            if (m.y < 0 || m.y > getHeight() - m.img.getHeight(null)) {

                m.speed \*= -1;

            }

        }

    }

    class MyPanel extends JPanel {

        @Override

        protected void paintComponent(Graphics g) {

            super.paintComponent(g);

            for (Monster m : monsters) {

                m.draw(g);

            }

        }

    }

}

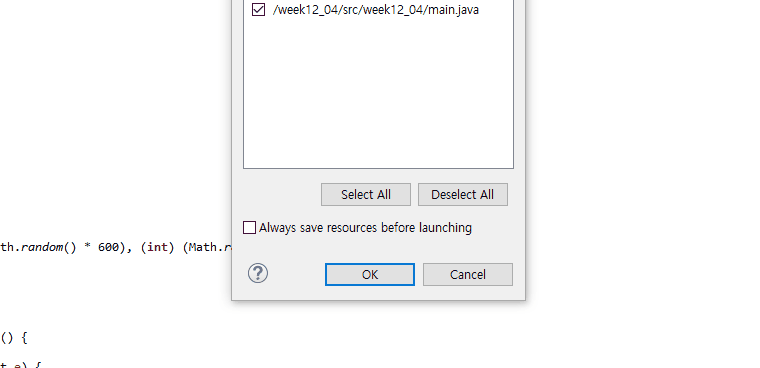
public class main {

    public static void main(String[] args) {

        MyFrame f = new MyFrame();

    }

}



이 프로그램은 10개의 몬스터를 랜덤생성하고, 몬스터를 클릭한 이후 모든 몬스터가 사라지면 자동으로 프로그램을 종료하는 기능을 한다. 우선 이 몬스터들은 monsters라는 이름의 ArrayList에 저장이 된다. for문을 통해 10개의 monster 클래스 객체를 monsters에 저장한다. 만약 클릭이 발생했을 경우, 모든 객체의 x, y값을 가져와 객체의 크기 내에 존재한다면 해당 객체를 삭제한다. 비교가 끝나면 다시 repaint를 통해 객체를 삭제한다. 만약 monsters의 크기가 0 즉슨 모든 몬스터가 사라졌다면 시스템을 종료한다.

해당 프로그램은 타이머를 통해 작동한다. 0.1초에 한번씩 작동하는 타이머를 세팅하고, 해당 타이머가 작동할 시, 객체들을 움직이도록 move를 하고, repaint를 통해 객체들을 움직이게 한다. Move에서는 몬스터들을 움직이도록 한다. 지정된 speed만큼 x와 y에 더해줌으로써 객체를 움직인다. 만약 x, y 좌표가 스크린의 영역을 벗어나면 speed에 -1을 곱해 반대방향으로 움직일 수 있도록 한다.