실력 상승! 연습 문제

Q1 다음은 서로 다른 2가지 방법으로 List 객체를 생성한 후 데이터를 추가한 예다. 오류가 발생한 위치와 그 원인을 설명하시오.

```
01 public static void main(String[] args) {
02     List<Integer> list1 = new ArrayList<Integer>();
03     list1.add(1);
04     list1.add(2);
05     list1.add(3);
06     System.out.println(list1);
07     List<Integer> list2 = Arrays.asList(1, 2);
08     list2.add(3);
09     System.out.println(list2);
10 }
```

실행 결과

```
[1, 2, 3]
[1, 2]
```

오류가 발생한 행 번호	오류가 발생한 원인
8	asList(.)로 리스트 객체를 생성하는 경우 데이터 길이의 변경이 불가함

Q2 다음은 ArrayList 생성자를 이용해 List 객체를 생성하고 add() 메서드와 remove() 메서드를 이용해 데이터를 추가, 삭제한 코드다. 다음 코드의 실행 결과를 쓰시오.

```
public static void main(String[] args) {
    List<Integer> list = new ArrayList<>();
    list.add(2);
    list.add(3);
    list.add(4);
    System.out.println(list);
    list.remove(2);
    System.out.println(list);
}
```

실행 결과

```
[2, 3, 4]
[2, 3]
```

Q3 다음은 Vector의 생성자를 이용해 List의 객체를 생성한 예다. 다음 코드의 실행 결과를 쓰시오.

```
public static void main(String[] args) {
    List<Boolean> list = new Vector<>();
    list.add(true);
    list.add(false);
    list.add(true);
    Boolean[] bArray = list.toArray(new Boolean[5]);
    System.out.println(Arrays.toString(bArray));
}
```

실행 결과

```
[true, false, true, null, null]
```

Q4 다음은 List 인터페이스를 구현한 자식 클래스의 생성자를 이용해 List 객체를 생성한 후 0번 위치에 100,000개의 데이터를 추가하는 코드다. 이때 빈칸에 들어갈 3개의 자식 클래스(ArrayList, LinkedList, Vector) 중에서 가장 효율적인 자식 클래스의 생성자를 선택하고 그 이유를 설명하시오.

```
public static void main(String[] args) {
    List<String> list = new          LinkedList<String>()
    for(int i = 0; i < 100000; i++) {
        list.add(0, i + "데이터");
    }
    System.out.println("완료");
}
```

이유:
LinkedList는 좌우 데이터와의 연결정보만을 저장하기 때문에 데이터의 추가 삭제 속도가 다른 List 구현클래스보다 빠름

실행 결과

```
완료
```

Q5 다음과 같이 클래스 Data가 정의돼 있다.

```
class Data {
    int m;
    public Data(int m) {
        this.m = m;
    }
    @Override
    public boolean equals(Object obj) {
        if(obj instanceof Data)
            return this.m == ((Data)obj).m;
        else
            return false;
    }
}
```

다음 코드의 실행 결과를 쓰시오.

```
public static void main(String[] args) {
    Data data1 = new Data(3);
    Data data2 = new Data(3);
    System.out.println(data1 == data2);
    System.out.println(data1.equals(data2));
    System.out.println(data1.hashCode() == data2.hashCode());
}
```

실행 결과

```
false
true
false
```

Q6 다음과 같이 클래스 Data가 정의돼 있다.

```
class Data{
    int m;
    public Data(int m) {
        this.m = m;
    }
    @Override
    public boolean equals(Object obj) {
        if(obj instanceof Data)
            return this.m == ((Data)obj).m;
        else
            return false;
    }
    @Override
    public int hashCode() {

        return m;
        //또는 return Objects.hash(m);
        //또는 Integer(m).hashCode();

    }
}
```

다음 코드의 실행 결과로 2가 나오도록 클래스 Data 내에 hashCode() 메서드의 내부를 작성하시오.

```
public static void main(String[] args) {
    Set<Data> set = new HashSet<>();
    set.add(new Data(2));
    set.add(new Data(2));
    set.add(new Data(3));
    System.out.println(set.size());
}
```

실행 결과

```
2
```

Q7 다음 MyData 클래스는 크기를 비교하기 위해 Comparable<MyData> 인터페이스를 구현했다. 또한 toString() 메서드를 필드인 str을 리턴하도록 오버라이딩했다. 이때 다음과 같은 실행 결과(글자 수의 오름차순)가 나오도록 compareTo 메서드의 내부를 작성하시오.

```
class MyData implements Comparable<MyData> {
    String str;
    public MyData(String str) {
        this.str = str;
    }
    @Override
    public int compareTo(MyData o) {

        if(str.length()<o.str.length())
            return -1;
        else if (str.length()==o.str.length())
            return 0;
        else
            return 1;

    }
    @Override
    public String toString() {
        return str;
    }
}

public static void main(String[] args) {
    MyData md1 = new MyData("자바 프로그램");
    MyData md2 = new MyData("반가워");
    MyData md3 = new MyData("감사합니다");
    TreeSet<MyData> treeSet = new TreeSet<>();
    treeSet.add(md1);
    treeSet.add(md2);
    treeSet.add(md3);
    System.out.println(treeSet);
}
```

실행 결과

```
[반가워, 감사합니다, 자바 프로그램]
```

Q8 다음은 HashMap을 이용해 Map 객체를 생성한 후 (Key, Value)의 쌍을 추가한 코드다. 빈칸에 들어갈 코드와 실행 결과를 쓰시오(단, 실행 결과는 Key = Value, Key = Value, … 의 형식으로 작성하며 순서는 상관없음).

```
public static void main(String[] args) {
    Map<String, Boolean> map = new HashMap<>();
    map.        put        ("사운드", true);
    map.        put        ("그래픽", false);
    map.        put        ("배경음", true);
    map.        put        ("그래픽", true);

    System.out.println(map);
}
```

실행 결과

```
{배경음=true, 사운드=true, 그래픽=true}
```

Q9 다음은 Stack 객체를 생성하고 데이터 입출력 메서드와 위치 확인 메서드를 활용한 코드다. 실행 결과를 쓰시오.

```
public static void main(String[] args) {
    Stack<Double> stack = new Stack<Double>();
    stack.push(1.1);
    stack.push(2.2);
    stack.pop();
    stack.push(3.3);
    stack.push(4.4);

    System.out.println(stack.search(1.1));
    System.out.println(stack.search(2.2));
    System.out.println(stack.search(3.3));
    System.out.println(stack.search(4.4));
}
```

실행 결과

```
3
-1
2
1
```

Q10 다음은 LinkedList 생성자를 이용해 Queue 객체를 생성한 후 데이터의 입출력을 수행한 코드다. 다음 실행 코드의 출력 결과를 쓰시오.

```
public static void main(String[] args) {
    Queue<String> queue = new LinkedList<>();

    queue.offer("땡큐");
    queue.offer("베리");
    queue.offer("감사");
    queue.poll();
    queue.offer("방가");
    System.out.println(queue.peek());
    System.out.println(queue.poll());
    System.out.println(queue.poll());
    System.out.println(queue.poll());

}
```

실행 결과

```
베리
베리
감사
방가
```