**JAVA 실습 9주차 리포트**

20234014 컴퓨터 공학과 이은정

1. 덧셈 뺄셈 출력

**package** math;

**import** java.util.Scanner;

**public** **class** Calculator {

**public** **int** add(**int** a, **int** b) {

**return** a + b;

}

**public** **int** subtract(**int** a, **int** b) {

**return** a-b;

}

}

**package** main;

**import** math.Calculator;

**public** **class** Test **extends** Calculator {

**public** **static** **void** main(String[] args) {

Calculator calc = **new** Calculator();

**int** sum = calc.add(5, 3);

System.out.println("5 + 3 = " + sum);

**int** sub = calc.subtract(5, 3);

System.out.println("5 - 3 = " + sub);

}

}



각각 math, main 이라는 다른 패키지에서 함수가 정의가 되어 있다. Default 접근 지정자를 사용해도, 패키지가 다르면 서로 메서드를 사용할 수 없다. 그러나 extends를 사용해 클래스를 상속할 시, 패키지가 달라도 메서드를 사용할 수 있다. Math 패키지의 Calculator 클래스에서 int a, int b를 더하고 빼는 메서드를 정의한다. 그리고 main 에서 해당 메서드를 사용하기 위해 Calculator 클래스를 상속한다. 이렇게 한다면, 다른 패키지 간에서도 메서드를 상속 받아 사용할 수 있다. 다만, 아예 패키지가 다르기 때문에 import를 통해 해당 클래스를 불러와야 한다. Math 클래스의 Calculator 클래스를 불러와야 하므로, **import** math.Calculator; 를 통해 해당 클래스를 불러와 사용한다.

2. **class** Student {

**private** String studentID;

**private** String name;

Student(String ID, String name) {

studentID = ID;

**this**.name = name;

}

**public** String getStudentID() {

**return** studentID;

}

**public** **void** setStudentID(String studentID) {

**this**.studentID = studentID;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **boolean** equals(Object obj) {

**if** (obj **instanceof** Student) {

Student otherStudent = (Student) obj;

**return** **this**.studentID.equals(otherStudent.studentID);

}

**return** **false**;

}

}

**public** **class** StudentTest {

**public** **static** **void** main(String[] args) {

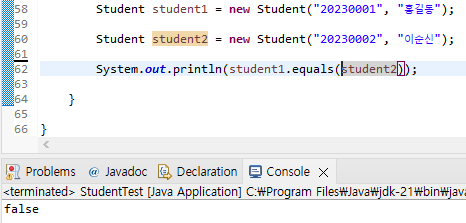
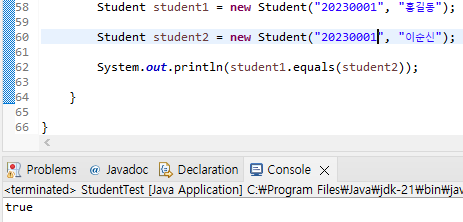
Student student1 = **new** Student("20230001", "홍길동");

Student student2 = **new** Student("20230002", "이순신");

System.***out***.println(student1.equals(student2));

}

}



ㄹ

다음은 main, Student 클래스 2개로 구성된 코드이다. Student 클래스에서는 StudentID, name을 매개변수로 받아 저장한다. main에서 학번, 이름을 매개변수로 주어 2개의 객체를 만든다. Student 클래스에는 접근자, 설정자 뿐만 아니라 다른 객체와 학번을 비교할 수 있는 메서드인 equals가 있다. Java.lang.Object 클래스에서 가지고 있던 메서드를 오버라이딩 하는 것이므로, public Boolean 형을 그대로 사용해 오버라이딩 한다. 먼저 이 코드에서는 Student 클래스를 정의 하였으나, 오버라이딩을 하게 되면 매개변수를 변경시킬 수 없으므로, 받은 객체가 Student 객체인지 확인하는 작업이 필요하다 (obj **instanceof** Student) 를 통해 입력받은 obj 객체가 Student 클래스인지 확인한다. 만약 Student 클래스라면, 명시적으로 다운캐스팅을 해준다. 이후, 이 객체와 타 클래스의 객체가 같은지 equals를 통해 비교해준다. 비교 결과를 return 하면 equals를 제대로 오버라이딩할 수 있게 된다.

3.

**package** math2;

**interface** Calculator{

**int** calculator(**int** x, **int** y);

}

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Calculator addition = **new** Calculator() {

@Override

**public** **int** calculator(**int** x, **int** y) {

**return** x+y;

}

};

**int** result = addition.calculator(5, 3);

System.***out***.println("결과 = "+result);

}

}

덧셈을 익명 클래스로 구현한다. 인터페이스 Calculator 을 만들고 난 이후, 추상 메서드 calculator을 그 안에 만든다. Calculator 객체를 하나 만들 때, Calculator은 인터페이스이므로, 메서드를 오버라이딩하여 완전하게 만들어 주어야 한다. 따라서 calculator을 오버라이딩 하여 int x, int y를 return 하도록 코드를 수정해준다. 이후, addition 객체를 통해 calculator 메서드를 사용할 수 있게 된다.



4.

**package** math2;

**interface** Queue {

**void** enqueue(**int** item);

**int** dequeue();

**boolean** isEmpty();

}

**class** MyQueue **implements** Queue {

**private** **int** MaxSize;

**private** **int**[] queueArray;

**private** **int** front;

**private** **int** rear;

**private** **int** currentSize;

**public** MyQueue() {

MaxSize = 5;

currentSize = 0;

front = 0;

rear = 0;

queueArray = **new** **int**[MaxSize];

}

**public** **int** getFront() {

**return** front;

}

**public** **void** setFront(**int** front) {

**this**.front = front;

}

**public** **int** getRear() {

**return** rear;

}

**public** **void** setRear(**int** rear) {

**this**.rear = rear;

}

@Override

**public** **void** enqueue(**int** item) {

**if** (isfull() == **false**) {

rear= (rear+1+MaxSize)%MaxSize;

queueArray[rear] = item;

currentSize++;

}

**else** System.***out***.println("큐가 찼습니다.");

}

@Override

**public** **int** dequeue() {

**if** (isEmpty() == **false**) {

front= (front+1+MaxSize)%MaxSize;

currentSize--;

**return** queueArray[(front+MaxSize)%MaxSize];

}

**else** {

System.***out***.println("큐가 비었습니다.");

**return** -1;

}

}

@Override

**public** **boolean** isEmpty() {

**if** (rear == front)

**return** **true**;

**else** {

**return** **false**;

}

}

**boolean** isfull() {

**if** ((rear + 1 + MaxSize) % MaxSize == front)

**return** **true**;

**else** {

**return** **false**;

}

}

}

**public** **class** QueueTest {

**public** **static** **void** main(String[] args) {

MyQueue q = **new** MyQueue();

**int** i=1;

**do**{

q.enqueue(i);

System.***out***.println("rear: "+ q.getRear()+"\tfront: "+q.getFront());

i++;

}**while**(i<5);

System.***out***.println("\nrear: "+ q.getRear()+"\tfront: "+q.getFront()+"\tVal: "+ q.dequeue());

System.***out***.println("rear: "+ q.getRear()+"\tfront: "+q.getFront()+"\tVal: "+ q.dequeue());

System.***out***.println("rear: "+ q.getRear()+"\tfront: "+q.getFront()+"\tVal: "+ q.dequeue());

System.***out***.println("\n");

i=1;

**do**{

System.***out***.println("rear: "+ q.getRear()+"\tfront: "+q.getFront());

q.enqueue(i);

i++;

}**while**(i<5);

System.***out***.println("\n");

System.***out***.println("rear: "+ q.getRear()+"\tfront: "+q.getFront()+"\tVal: "+ q.dequeue());

System.***out***.println("rear: "+ q.getRear()+"\tfront: "+q.getFront()+"\tVal: "+ q.dequeue());

System.***out***.println("rear: "+ q.getRear()+"\tfront: "+q.getFront()+"\tVal: "+ q.dequeue());

}

}

큐에는 enqueue, dequeue, isEmpty 메서드가 있다. Enqueue는 큐에 값을 집어 넣는 메서드이고, dequeue는 큐에서 값을 빼내는 메서드이고, isFull은 큐가 가득 찼는지 검사하는 메서드이다. 위 큐 메서든 인터페이스에서 추상 메서드로 구현하고 MyQueue에서 해당 인터페이스를 상속 받아 재정의하여 사용한다. 큐에는 최대 사이즈, front, rear이라는 변수가 필요하다. 또한 값을 저장할 리스트인 queueArray를 만들고, 큐의 값 개수를 셀 currentSize 인스턴스 변수 또한 만든다. main에서 MyQueue 객체를 만들 때, 매개변수를 통해 MaxSize 값을 전달해줘도 되고, 생성자를 통해 내가 임의로 값을 정해주어도 좋다. 여기에서는 임의로 생성자를 통해 MaxSize 크기를 5로 정해보도록 하겠다. CurrentSize, front, rear 는 0으로 초기화 해준다.

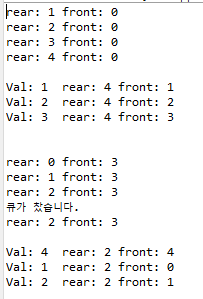
먼저 이후로 계속 사용할 isEmpty, isFull 메서드에 대해 먼저 설명하자. 만약 front==rear이라면 해당 큐는 전부 비었다고 간주한다. isFull는 rear + 1 + MaxSize) % MaxSize == front 일 때 가득 찼다고 간주한다.

enqueue는 int n이라는 매개변수를 가진다. Enqueue는 값을 저장하는 메서드이다. 우선 큐가 가득 차지 않아야만 값을 저장할 수 있으므로, if(isFull==false) 일 때만 값을 저장하는 기능을 수행한다. 만약 isFull==False라면 rear= (rear+1+MAX)%max; queueArray[rear] = n;를 통해 작동한다.

Dequeue는 값을 반환하는 메서드이다. 큐가 비지 않았을 때, front를 1 증가시켜 그 위치의 값을 반환한다. 따라서 isEmpty() == **false** 일 때, front = (front + 1 + MaxSize) % MaxSize; return queueArray[(front + MaxSize) % MaxSize]; 를 통해 작동한다. Enqueue, dequeue는 각각 currentSize를 변화시키므로, 해당 인스턴스의 수도 변화시키도록 한다.

다음은 main함수이다. q라는 이름의 객체를 하나 생성하고, 여기에 값을 넣고 빼는 동작을 하고 있다.



위 코드에 따르면, 5개의 공간이 있기 때문에 4개의 값을 저장할 수 있다. 오른 쪽의 결과를 보면 4번 반복을 하는 동안, rear이 1씩 증가하며 제대로 저장되는 모습을 볼 수 있다. Dequeue를 통해 값을 제거하면 값이 차례차례 출력되고, front 값이 1씩 오르는 것을 볼 수 있다.

값을 처음으로 dequeue하고 나면 4번 인덱스에 4 하나만 남아있는 상태이다. 즉슨 우리는 3개의 값을 저장할 수 있다. 다시 enqueue를 4번 반복해보자. Rear이 0으로 초기화 되어 다시 값이 저장되다가 rear이 2가 되고 나자 더 이상 값을 저장하지 않는 모습을 볼 수 있다. 이후에 dequeue를 통해 값을 출력하면 원형 큐의 정의에 따라 제대로 작동하는 모습을 볼 수 있다.