Assignment(5~8)

Basic School(#5)

Program Output

- Make a Java program that can manage students for a school

```
Enter Command String! add
James 1
Enter Command String! list
School Name: PNU Student Count: 1
      [James, 1학년]

Enter Command String! add
Brown 2
Enter Command String! list
School Name: PNU Student Count: 2
      [James, 1학년]
      [Brown, 2학년]
```

```
Enter Command String! find
Brown 2
[Brown, 2학년]
Enter Command String! find
Brown 1
Student Not Found with name Brown and year 1
Enter Command String! add
Kim 4
Enter Command String! list
School Name: PNU Student Count: 3
      [James, 1학년]
      [Brown, 2학년]
      [Kim, 4학년]

Enter Command String! clear
Enter Command String! list
School Name: PNU Student Count: 0
```

SchoolTest.java

```
import java.util.Scanner;

enum OperationKind{
  ADD,
  FIND,
  CLEAR,
  LIST,
  INVALID,
  QUIT
};

public class SchoolTest {
  private static Scanner scanner = new Scanner(System.in);

  public static void main(String[] args) {
    School pnu = new School("PNU", 100);
    while (true) {
      final OperationKind op = getOperation();
      if (op == OperationKind.QUIT) {
        System.out.println("Bye");
        break;
      }
      if (op == OperationKind.INVALID) {
        System.out.println("Invalid Operaion!");
        break;
      }

      switch(op) {
      case ADD: {
        Student newStudent = createStudent();
        pnu.addStudent(newStudent);
        break;
      }
      case FIND: findStudent(pnu); break;
      case CLEAR: pnu.removeAllStudent(); break;
      case LIST: System.out.println(pnu); break;
      default: break;
      }
    }
  }

  // getOperation
  public static OperationKind getOperation() {
    System.out.print("Enter Command String! ");
    String cmd = scanner.next();
    cmd = cmd.toUpperCase();

    OperationKind command;
```

```
    switch (cmd) {
    case "ADD":
      command = OperationKind.ADD;
      break;
    case "FIND":
      command = OperationKind.FIND;
      break;
    case "CLEAR":
      command = OperationKind.CLEAR;
      break;
    case "LIST":
      command = OperationKind.LIST;
      break;
    case "QUIT":
      command = OperationKind.QUIT;
      break;
    default:
      command = OperationKind.INVALID;
      break;
    }
    return command;
  }

  // createStudent
  public static Student createStudent() {
    final String studenntName = scanner.next();
    final int schoolYear = scanner.nextInt();

    return  new Student(studenntName, schoolYear);
  }

  // findStudent
  public static void findStudent(final School school) {
    final String studentName = scanner.next();
    final int schoolYear = scanner.nextInt();
    final Student foundStudent = school.findStudent(studentName, schoolYear);

    if(foundStudent != null)
      System.out.println(foundStudent);
    else
      System.out.println("Student Not Found with name " + studentName + " and year " + schoolYear);
  }
}
```

School.java

```
import java.util.Arrays;
import java.util.Objects;

public class School {
  private String name;
  private int limit;

  private Student[] students;
  private int studentCount = 0;

  public School(final String name, final int limit) {
    this.name = name;
    this.limit = limit;
    students = new Student[this.limit];
  }

  public int getStudentCount() {
    return studentCount;
  }

  public String toString() {
    String msg = "School Name: " + name + " Student Count: " + studentCount + "\n";

    for(int i = 0; i < studentCount; i++) {
      msg += "\t" + students[i] + "\n";
    }
    return msg;
  }

  public void addStudent(Student newStudent) {
    if (studentCount < limit) {
```

```
      students[studentCount] = newStudent;
      studentCount++;
    }
  }

  public Student findStudent(String studentName, int schoolYear) {
    for (final Student student : students) {
      if (student == null) return null;
      if (student.match(studentName, schoolYear))
        return student;
    }
    return null;
  }

  public void removeAllStudent() {
    for (int i = 0; i < studentCount; i++) {
      students[i] = null;
    }
    studentCount = 0;
  }

  @Override
  public int hashCode() {
    return Objects.hash(name, limit, students, studentCount);
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null) return false;
    if (getClass() != obj.getClass()) return false;
    School other = (School) obj;

    return Objects.equals(name, other.name) && limit == other.limit && studentCount == other.studentCount && Arrays.equals(students, other.
  }
}
```

Student.java

```
import java.util.Objects;

public class Student {
  private final String name;
  private int year;

  public Student(String studentName, int schoolYear) {
    assert studentName != null;
    assert schoolYear >= 1 && schoolYear <= 4;

    name = studentName;
    year = schoolYear;
  }

  public int getYear() {
    return year;
  }

  public void setYear(int year) {
    this.year = year;
  }

  public String getName() {
    return name;
  }

  public boolean match(String studentName, int schoolYear) {
    return Objects.equals(name, studentName) && year == schoolYear;
  }

  @Override
  public String toString() {
    return "Student [ name=" + name + ", year=" + year + "학년 ]";
  }

  @Override
  public boolean equals(Object otherObject) {
    if (this == otherObject) return true;
```

```
    if (otherObject == null) return false;
    if (getClass() != otherObject.getClass()) return false;
    Student other = (Student) otherObject;

    return Objects.equals(name, other.name) && year == other.year;
  }
}
```

SchoolManager(#6)

```
Enter Operation String! add
SNU Brown 2
[Name: Brown, School:SNU, 2학년]
Enter Operation String! list
Total School Count: 2
School Name: PNU Student Count: 1
[Name: James, School:PNU, 1학년]

School Name: SNU Student Count: 2
[Name: James, School:SNU, 1학년]
[Name: Brown, School:SNU, 2학년]

Enter Operation String! find
James 1
2 found
[Name: James, School:PNU, 1학년]
[Name: James, School:SNU, 1학년]
Enter Operation String! find
James 2
No Student Found with name James and year 2
Enter Operation String! Clear
Enter Operation String! list
Total School Count: 0
```

SchoolManager.java

```
import java.util.Scanner;
import java.util.List;

enum OperationKind{
  ADD,
  FIND,
  CLEAR,
  LIST,
  INVALID,
  QUIT
};

public class SchoolManagerTest {
  private static Scanner scanner = new Scanner(System.in);
  private static SchoolManager schoolManager = new SchoolManager();

  public static void main(String[] args) {
    while (true) {
      final OperationKind op = getOperation();

      if (op == OperationKind.QUIT) {
        System.out.println("Bye");
        break;
      }
      if (op == OperationKind.INVALID) {
        System.out.println("Invalid Operaion!");
        break;
      }

      switch (op) {
      case ADD: {
        Student newStudent = createStudent();
        System.out.println(newStudent);
        break;
      }
      case FIND:
        findStudent(); break;
      case CLEAR:
        schoolManager.removeAllSchools(); break;
      case LIST:
        System.out.println(schoolManager); break;
      default: break;
      }
    }
```

```
  }

  // getOperation
  public static OperationKind getOperation() {
    System.out.print("Enter Command String! ");
    String cmd = scanner.next();
    cmd = cmd.toUpperCase();

    OperationKind command;
    switch (cmd) {
    case "ADD":
      command = OperationKind.ADD;
      break;
    case "FIND":
      command = OperationKind.FIND;
      break;
    case "CLEAR":
      command = OperationKind.CLEAR;
      break;
    case "LIST":
      command = OperationKind.LIST;
      break;
    case "QUIT":
      command = OperationKind.QUIT;
      break;
    default:
      command = OperationKind.INVALID;
      break;
    }
    return command;
  }

  private static Student createStudent() {
    final String schoolName = scanner.next();
    final String studentName = scanner.next();
    final int schoolYear = scanner.nextInt();

    School theSchool = schoolManager.findSchool(schoolName);
    if (theSchool == null)
      theSchool = schoolManager.createSchool(schoolName);
    final Student newStudent = new Student(theSchool, studentName, schoolYear);
    theSchool.addStudent(newStudent);

    return newStudent;
  }

  private static void findStudent() {
    final String studentName = scanner.next();
    final int schoolYear = scanner.nextInt();
    final List<Student> foundStudents = schoolManager.findStudent(studentName, schoolYear);

    if (foundStudents.size() > 0) {
      System.out.println(foundStudents.size() + " found");
      for (Student s : foundStudents) System.out.println(s);
    }
    else
      System.out.println("No Student Found with name " + studentName + " and year " + schoolYear);
  }
}
```

SchoolManager.java

```
import java.util.List;
import java.util.ArrayList;

public class SchoolManager {
  private List<School> schools = new ArrayList<>();
  private int schoolCount = 0;

  // method
  public int getCount() {
    return schoolCount;
  }

  public School findSchool(String schoolName) {
    for (final School school : schools) {
      if (school == null) return null;
```

```
      if (school.match(schoolName))
        return school;
    }
    return null;
  }

  public School createSchool(String schoolName) {
    School newSchool = new School(schoolName);
    schools.add(newSchool);
    schoolCount++;
    return newSchool;
  }

  public void removeAllSchools() {
    schools.clear();
    schoolCount = 0;
  }

  public List<Student> findStudent(String studentName, int schoolYear) {
    List<Student> foundStudent = new ArrayList<>();
    for (final School school : schools) {
      List<Student> tmp = school.getStudents();
      for (final Student student : tmp) {
        if (student.match(school, studentName, schoolYear)) {
          foundStudent.add(student);
        }
      }
    }

    return foundStudent;
  }

  @Override
  public String toString() {
    String msg = "Total School Count: " + this.getCount() + "\n";
    for (final School school : schools) {
      String schoolMsg = "School Name: " + school.getName() + " Student Count: " + school.getStudentCount() + "\n";
      msg += schoolMsg;

      List<Student> stringStudent = school.getStudents();
      for (final Student student : stringStudent) {
        String tmp = student.toString() + "\n";
        msg += tmp;
      }
      msg += "\n";
    }
    return msg;
  }
}
```

School.java

```
import java.util.Objects;
import java.util.List;
import java.util.ArrayList;

public class School {
  private String name;
  private List<Student> students = new ArrayList<>();

  // getter
  public String getName() {
    return name;
  }

  public List<Student> getStudents() {
    return students;
  }

  public int getStudentCount() {
    return students.size();
  }

  // generator
  public School(final String name) {
    this.name = name;
  }
```

```
  // method
  public void addStudent(Student newStudent) {
    students.add(newStudent);
  }

  public boolean match(String schoolName) {
    return Objects.equals(name, schoolName);
  }

  @Override
  public int hashCode() {
    return Objects.hash(name, students);
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null) return false;
    if (getClass() != obj.getClass()) return false;
    School other = (School) obj;

    return Objects.equals(name, other.name) && students.equals(other.students);
  }
}
```

Student.java

```
import java.util.Objects;

public class Student {
  private final String name;
  private int year;
  private final School theSchool;

  // generator
  public Student(School theSchool, String name, int year) {
    this.name = name;
    this.year = year;
    this.theSchool = theSchool;
  }

  // method
  public boolean match(School schoolName, String studentName, int schoolYear) {
    return Objects.equals(theSchool, schoolName) && Objects.equals(name, studentName) && year == schoolYear;
  }

  @Override
  public String toString() {
    return "[ Name: " + name + ", School: " + theSchool.getName() + ", " + year +"학년 ]";
  }

  @Override
  public boolean equals(Object otherObject) {
    if (this == otherObject) return true;
    if (otherObject == null) return false;
    if (getClass() != otherObject.getClass()) return false;
    Student other = (Student) otherObject;

    return Objects.equals(theSchool, other.theSchool) && Objects.equals(name, other.name) && year == other.year;
  }

}
```

Shape Package(#7)

Sample Output

```
Enter Operation String! addr
10 10
[ Rectangle 10 10 100.000000]
Enter Operation String! addc
20 10 10
[ Circle [20, 10] 10 314.000000]
Enter Operation String! list
[[ Rectangle 10 10 100.000000], [ Circle [20, 10] 10 314.000000]]
Enter Operation String! addr
5 10
[ Rectangle 5 10 50.000000]
Enter Operation String! addr 10 40
[ Rectangle 10 40 400.000000]
Enter Operation String! list
[[ Rectangle 10 10 100.000000], [ Circle [20, 10] 10 314.000000],
[ Rectangle 5 10 50.000000], [ Rectangle 10 40 400.000000]]
Enter Operation String! clear
Enter Operation String! list
[]
Enter Operation String! quit
Bye
```

EditorTest.java

```
import java.util.Scanner;

enum OperationKind {
  ADDR,
  ADDC,
  LIST,
  CLEAR,
  QUIT,
  INVALID
};

public class EditorTest {
  private static Editor editor = new Editor() ;

  public static void main(String[] args) {
    final Scanner scanner = new Scanner(System.in);
    while ( true ) {
      final OperationKind op = getOperation(scanner) ;
      if ( op == OperationKind.QUIT ) {
        System.out.println("Bye") ; break;
      }
      if ( op == OperationKind.INVALID ) {
        System.out.println("Invalid Operation!") ; continue ;
      }
      switch ( op ) {
      case ADDR : {
        final Rectangle newRectangle = createRectangle(scanner) ;
        System.out.println(newRectangle) ;
        editor.add(newRectangle) ; break ;
      }
      case ADDC : {
        final Circle newCircle = createCircle(scanner) ;
        System.out.println(newCircle) ;
        editor.add(newCircle) ; break ;
      }
      case CLEAR: editor.clear() ; break ;
      case LIST:  editor.list() ; break ;
      default: break;
      }
    }
    scanner.close();
  }

  private static Circle createCircle(final Scanner scanner) {
    final int x = scanner.nextInt() ;
    final int y = scanner.nextInt() ;
    final int radius = scanner.nextInt() ;

    final Circle newCircle = new Circle(new Point(x, y), radius) ;
    return newCircle ;
  }

  private static Rectangle createRectangle(final Scanner scanner) {
    final int width = scanner.nextInt() ;
    final int height = scanner.nextInt() ;
```

```
    final Rectangle newRectangle = new Rectangle(width, height) ;
    return newRectangle ;
  }

  private static OperationKind getOperation(final Scanner scanner) {
    System.out.print("Enter Operation String! ") ;
    final String operation = scanner.next() ;

    OperationKind kind;
    try {
      kind = OperationKind.valueOf(operation.toUpperCase());
    }
    catch ( IllegalArgumentException e ) {
      kind = OperationKind.INVALID;
    }
    return kind;
  }
}
```

Editor.java

```
import java.util.List;
import java.util.ArrayList;

public class Editor {
  private List<Object> shapes = new ArrayList<>();

  // method
  public void add(Object newObject) {
    shapes.add(newObject);
  }

  public void clear() {
    shapes.clear();
  }

  public void list() {
    String msg = "[";
    for (final Object shapeOne: shapes) {
      msg += shapeOne.toString();
      msg += ", ";
    }
    msg += "]";

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

Rectangle.java

```
public class Rectangle {
  private int width, height;

  public Rectangle(int width, int height) {
    super();
    this.width = width;
    this.height = height;
  }

  @Override
  public String toString() {
    return "[ Rectangle " + this.width + " " + this.height + " " + width*height + " ]";
  }
}
```

Point.java

```
public class Point {
  private int x, y;

  public Point(int x, int y) {
```

```
    super();
    this.x = x;
    this.y = y;
  }

  @Override
  public String toString() {
    return "[ " + this.x + ", " + this.y + " ]";
  }
}
```

Circle.java

```
public class Circle {
  private Point center;
  private int radius;
  public Circle(Point center, int radius) {
    super();
    this.center = center;
    this.radius = radius;
  }

  @Override
  public String toString() {
    return "[ Circle " + this.center.toString() + " " + radius + " " + radius*radius*Math.PI + " ]";
  }
}
```

Inheritance(#8)

입력	출력
ADD R 10 20	Rectangle 10 20 200.0
ADD C 10 10 10	Circle 10 10 314.1592
Add R 20 20	Rectangle 20 20 400.0
PRINTALL	Rectangle 10 20 200.0 Circle 10 10 314.1592 Rectangle 20 20 400.0
PRINT t	NONE
Add T 20 20	Triangle 20 20 200.0
PRINTALL	Rectangle 10 20 200.0 Circle 10 10 314.1592 Rectangle 20 20 400.0 Triangle 20 20 200.0
PRINT r	Rectangle 10 20 200.0 Rectangle 20 20 400.0
Add r 20 20	Rectangle 20 20 400.0
PRINT C	Circle 10 10 314.1592
REMOVEALL	5
TOTALAREA	0
PRINT r	NONE
Add T 20 20	Triangle 20 20 200.0
Add T 20 20	Triangle 20 20 200.0
PRINT t	Triangle 20 20 200.0 Triangle 20 20 200.0
TOTALAREA	400.0
REMOVEALL	2
TOTALAREA	0

EditorTest.java

```
import java.util.Scanner;
enum OperationKind {ADD, PRINT, PRINTALL, REMOVEALL, TOTALAREA, QUIT, INVALID};
enum ShapeType {R, C, T};

public class EditorTest {
  private static Editor editor = new Editor();
  private static Scanner scanner = new Scanner(System.in);
  public static void main(String[] args) {
    while (true) {
      final OperationKind op = getOperation(scanner);
      if (op == OperationKind.QUIT) {
        System.out.println("Bye"); break;
      }
      if (op == OperationKind.INVALID) {
        System.out.println("Invalid Operation!"); continue;
      }
      switch (op) {
      case ADD : {
```

```
        ShapeType shapeOrder = getShapeType();
        if (shapeOrder == ShapeType.R) {
          final Rectangle newRectangle = createRectangle(scanner);
          System.out.println(newRectangle);
          editor.add(newRectangle); break;
        }
        else if (shapeOrder == ShapeType.C) {
          final Circle newCircle = createCircle(scanner);
          System.out.println(newCircle);
          editor.add(newCircle); break;
        }
        else if (shapeOrder == ShapeType.T) {
          final Triangle newTriangle = createTriangle(scanner);
          System.out.println(newTriangle);
          editor.add(newTriangle); break;
        }
        else break;
      }
      case PRINT : {
        ShapeType shapeOrder = getShapeType();
        if (shapeOrder == ShapeType.R) {
          editor.printR(); break;
        }
        else if (shapeOrder == ShapeType.C) {
          editor.printC(); break;
        }
        else if (shapeOrder == ShapeType.T) {
          editor.printT(); break;
        }
        else break;
      }
      case PRINTALL : editor.printall(); break;
      case REMOVEALL : editor.removeall(); break;
      case TOTALAREA : editor.totalarea(); break;
      default : break;
      }
    }
    scanner.close();
  }

  private static ShapeType getShapeType() {
    ShapeType shapeOrder;
    while(true){
      String shapeSearch = scanner.next();
      if(shapeSearch.equalsIgnoreCase("T")){
        shapeOrder = ShapeType.T;
        break;
      }
      else if(shapeSearch.equalsIgnoreCase("C")){
        shapeOrder = ShapeType.C;
        break;
      }
      else if(shapeSearch.equalsIgnoreCase("R")){
        shapeOrder = ShapeType.R;
        break;
      }
      else continue;
    }
    return shapeOrder;
  }

  private static Circle createCircle(final Scanner scanner) {
    final int x = scanner.nextInt();
    final int y = scanner.nextInt();
    final int radius = scanner.nextInt();

    final Circle newCircle = new Circle(new Point(x, y), radius);
    return newCircle;
  }

  private static Rectangle createRectangle(final Scanner scanner) {
    final int width = scanner.nextInt();
    final int height = scanner.nextInt();

    final Rectangle newRectangle = new Rectangle(width, height);
    return newRectangle;
  }

  private static Triangle createTriangle(final Scanner scanner) {
    final int width = scanner.nextInt();
    final int height = scanner.nextInt();
```

```
    final Triangle newTriangle = new Triangle(width, height);
    return newTriangle;
  }

  private static OperationKind getOperation(final Scanner scanner) {
    System.out.print("Enter Operation String!");
    final String operation = scanner.next();

    OperationKind kind;
    try {
      kind = OperationKind.valueOf(operation.toUpperCase());
    }
    catch (IllegalArgumentException e) {
      kind = OperationKind.INVALID;
    }
    return kind;
  }
}
```

Editor.java

```
import java.util.ArrayList;
import java.util.List;

public class Editor {
  private static ArrayList<Shape> ShapeList = new ArrayList<>();
  private static int shapeCount = 0;

  public static void add(Shape newShape) {
    ShapeList.add(newShape);
    shapeCount++;
  }

  public static void printall() {
    if(ShapeList.size()==0)
      System.out.println("NONE");
    for(int i=0; i<ShapeList.size();i++) {
      System.out.println(ShapeList.get(i));
    }
  }

  public static void printR() {
    for(int i=0; i<ShapeList.size(); i++) {
      if(ShapeList.get(i) instanceof Rectangle) {
        System.out.println(ShapeList.get(i));
      }
      else System.out.println("NONE");
    }
  }

  public static void printC() {
    for(int i=0; i<ShapeList.size(); i++) {
      if(ShapeList.get(i) instanceof Circle) {
        System.out.println(ShapeList.get(i));
      }
      else System.out.println("NONE");
    }
  }

  public static void printT() {
    for(int i=0; i<ShapeList.size(); i++) {
      if(ShapeList.get(i) instanceof Triangle) {
        System.out.println(ShapeList.get(i));
      }
      else System.out.println("NONE");
    }
  }


  public static void removeall() {
    System.out.println(ShapeList.size());
    ShapeList.clear();
    shapeCount = 0;
  }
```

```
  public static void totalarea() {
    float sum = 0;
    int intsum = 0;

    for(int i =0; i<ShapeList.size(); i++) {
      sum = sum + ShapeList.get(i).getSize();
    }
    if(sum == (int)sum) {
      intsum = (int)sum;
      System.out.println(intsum);
    }
    else
      System.out.println(sum);
  }
}
```

Shape.java

```
abstract public class Shape {
  abstract public String toString();

  abstract public float getSize();
}
```

Rectangle.java

```
public class Rectangle extends Shape{
  private int width = 0;
  private int height = 0;
  private float size = 0;

  public Rectangle(int width, int height) {
    this.width = width;
    this.height = height;
    this.size = (float) width * height;
  }

  @Override
  public String toString() {
    return "Rectangle " + width + " " + height + " " + size;
  }

  @Override
  public float getSize() {
    return size;
  }
}
```

Point.java

```
public class Point {
  public int x, y;

  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

Circle.java

```
public class Circle extends Shape {
  private Point center;
  private int radius = 0;
  private float size = 0;

  public Circle(Point center, int radius) {
    this.center = center;
```

```
    this.radius = radius;
    this.size = (float)(this.radius * this.radius * Math.PI);
  }

  @Override
  public String toString() {
    return "Circle " + center.x + " " + center.y + " " + size;
  }

  @Override
  public float getSize() {
    return size;
  }
}
```

Triangle.java

```
public class Triangle extends Shape{
  private int width = 0;
  private int height = 0;
  private float size = 0;

  public Triangle(int width, int height) {
    this.width = width;
    this.height = height;
    this.size = (float) width * height / 2;
  }

  @Override
  public String toString() {
    return "Triangle " + width + " " + height + " " + size;
  }

  @Override
  public float getSize() {
    return size;
  }
}
```