자바프로그래밍및실습 과제 4

214823 컴퓨터정보통신공학과 박종현

May 2022

코드 주석은 Javadoc 코드 문서화 스펙을 참조하여 작성함.
참조: https://docs.oracle.com/en/java/javase/17/docs/specs/javadoc/doc-comment-spec.html

1 과제 1

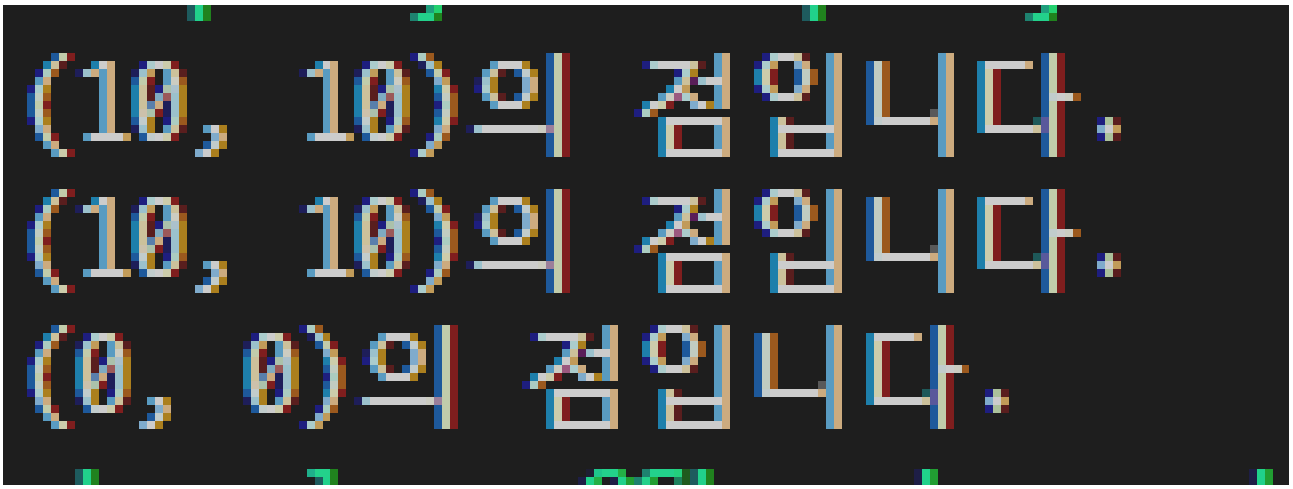
1.1 소스코드

```java
/**
 * @author @ShapeLayer
 */

// 문제 제시 코드
public class Main {
  public static void main(String[] args) {
    PositivePoint p = new PositivePoint();
    p.move(10, 10);
    System.out.println(p.toString() + "입니다.");

    p.move(-5, -5);
    System.out.println(p.toString() + "입니다.");

    PositivePoint p2 = new PositivePoint(-10, -10);
    System.out.println(p2.toString() + "입니다.");
  }
}

// 문제 제시 코드
class Point {
  private int x, y;
  public Point(int x, int y) { this.x = x; this.y = y; }
  public int getX() { return x; }
  public int getY() { return y; }
  protected void move(int x, int y) { this.x = x; this.y = y; }
}

/**
 * PositivePoint 클래스
 *
 * PositivePoint 클래스는 2차원 상의 한 점 중 1사분면과, 1사분면에 인접한 축 위에 있는 점에 대해
 ↪  표현합니다.
 */
public class PositivePoint extends Point {
  /**
   * PositivePoint 객체를 생성합니다.
   * @param {@coode void}
   */
  public PositivePoint() {
    super(0, 0);
  }

  /**
   * PositivePoint 객체를 생성합니다.
   * @param {@coode x} 점의 x좌표
   * @param {@coode y} 점의 y좌표
   */
  public PositivePoint(int x, int y) {
    super(0, 0);
    this.move(x, y);
  }

  /**
   * {@code setter}
```

```
55      * Point 클래스의 {@code move} 메서드를 오버라이드함
56      * 점을 특정 위치로 이동합니다.
57      *
58      * @param {@code x} 점의 x좌표
59      * @param {@code y} 점의 y좌표
60      */
61     @Override
62     protected void move(int x, int y) {
63       if (x < 0 || y < 0) return;
64       super.move(x, y);
65     }
66
67     /**
68      * 현재 표현하고 있는 점에 대한 정보를 문자열로 반환합니다.
69      * @return {@code String} 점 정보
70      */
71     public String toString() {
72       return String.format("(%d, %d)의 점", this.getX(), this.getY());
73     }
74   }
```

1.2 실행 예제

1.2.1 예제 1



2 과제 2

2.1 소스코드

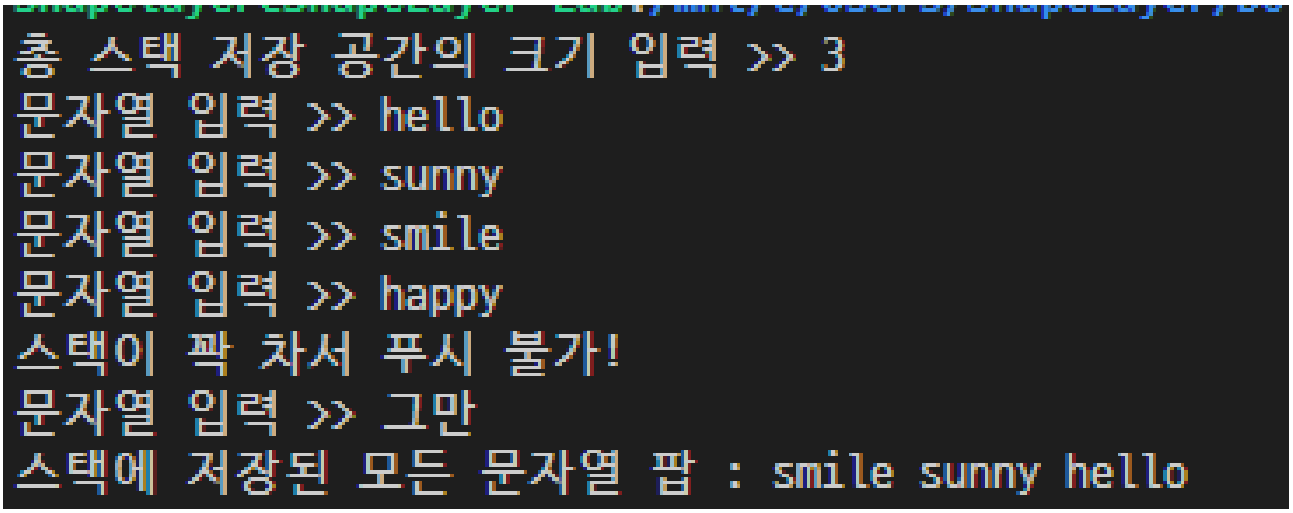
```
1   /**
2    * @author @ShapeLayer
3    */
4
5   import java.util.Collections;
6   import java.util.Scanner;
7
8   public class Main {
9     public static void main(String[] args) {
10      int n;
11      String gets;
12      StringStack stack;
13      Scanner sc = new Scanner(System.in); // 스캐너 객체 생성
14      System.out.print("총 스택 저장 공간의 크기 입력 >> ");
15      n = sc.nextInt();
```

```
    // 아래에서 정의한 StringStack 객체 생성
    stack = new StringStack(n);

    int i = 0;
    // "그만"이 입력될 때까지 문자열을 받아와야하므로 무한 루프 시작
    while (true) {
      System.out.print("문자열 입력 >> ");
      gets = sc.next();
      // 만약 "그만"이 입력됐다면 루프 종료
      if (gets.equals("그만")) break;
      // 스택에 입력값을 푸시함과 동시에 결과를 받아와서 성공했는지 여부 확인
      if (!stack.push(gets)) { // 만약 실패했다면 알림 출력
        System.out.println("스택이 꽉 차서 푸시 불가!");
      }
    }
    System.out.println("스택에 저장된 모든 문자열 팝 : " + String.join(" ", stack.getData()));
  }
}

// 문제 제시 인터페이스
interface Stack {
  int length();
  int capacity();
  String pop();
  boolean push(String val);
}

/**
 * StringStack 클래스
 * {@code Stack} 클래스 상속
 *
 * StringStack 클래스는 문자열 값을 스택 자료형으로 처리하는데 유용한 메서드가 포함되어 있습니다.
 */
class StringStack implements Stack {
  /**
   * 스택의 크기
   */
  int size
  /**
   * 스택의 가장 높은 값의 인덱스
   */
  int peek = -1;
  /**
   * 스택 데이터
   */
  String data[];

  /**
   * StringStack 객체를 생성합니다.
   * @param size {@code int} 스택의 크기
   */
  public StringStack(int size) {
    // 스택의 크기 기록 후 데이터 배열 초기화
    this.size = size;
    this.data = new String[this.size];
  }

  /**
   * 스택의 크기를 반환합니다.
   * @return {@code int} 스택의 크기
   */
  @Override
```

```
78    public int length() { return this.size; }
79
80    /**
81     * 스택의 용량을 반환합니다.
82     * @return {@code int} 스택의 용량
83     */
84    @Override
85    // peek는 가장 높은 값의 인덱스를 기록하므로, 용량을 표현하려면 1을 더해야함
86    public int capacity() { return peek+1; }
87
88    /**
89     * 스택에 값을 푸시합니다.
90     * 만약 성공했다면 {@code true}, 실패했다면 {@code false}를 반환합니다.
91     * @param push {@code String} 푸시할 값
92     */
93    @Override
94    public boolean push(String val) {
95      // 만약 스택의 가장 높은 값의 인덱스가 스택의 크기보다 작다면
96      // 즉, 스택에 잔여 공간이 있다면
97      if (this.peek < this.length()-1) {
98        // 꼭대기 인덱스 값을 높이고 값 업데이트
99        this.peek++;
100       this.data[this.peek] = val;
101       return true;
102     } else {
103       // 스택에 잔여 공간이 없다면 아무 일도 하지 않고 false 반환
104       return false;
105     }
106   }
107
108   /**
109    * 스택으로부터 값을 팝합니다.
110    * 만약 성공했다면 해당 값을 반환하고, 실패했다면 {@code null}을 반환합니다.
111    * @return {@code String?} 팝 결과
112    */
113   @Override
114   public String pop() {
115     // 만약 꼭대기 인덱스가 -1이라면 빈 스택이라는 의미이므로 null 리턴
116     if (this.peek < 0) return null;
117     // 아니라면 꼭대기 인덱스를 1 줄이고 결과 반환
118     // 값을 실제로 지우지는 않음: 새로 값을 푸시할 때 덮어쓰기함
119     this.peek--;
120     return this.data[this.peek+1];
121   }
122
123   /**
124    * 모든 데이터를 팝 했을 때 결과를 반환합니다.
125    * @return {@code String[]} 모든 데이터를 팝 했을 때 결과
126    */
127   public String[] getData() {
128     String[] results = new String[this.length()];
129     // 스택은 선입후출되므로, 데이터 배열을 그대로 반환하면 안됨
130     // O(N) 시간복잡도의 배열 reverse 처리
131     for (int i = 0; i < this.length(); i++) {
132       results[i] = this.data[this.length()-1-i];
133     }
134     return results;
135   }
136 }
```

2.2 실행 예제

2.2.1 예제 1



3 과제 3

3.1 소스코드

```java
/**
 * @author @ShapeLayer
 */

public class DictionaryApp {
  public static void main(String[] args) {
    Dictionary dic = new Dictionary(10);
    dic.put("황기태", "자바");
    dic.put("이재문", "파이선");
    dic.put("이재문", "C++");
    System.out.println("이재문의 값은 " + dic.get("이재문"));
    System.out.println("황기태의 값은 " + dic.get("황기태"));
    dic.delete("황기태");
    System.out.println("황기태의 값은 " + dic.get("황기태"));
  }
}

// 문제 제시 추상 클래스
abstract class PairMap {
  protected String keyArray [];
  protected String valueArray [];
  abstract String get(String key);
  abstract void put(String key, String value);
  abstract String delete(String key);
  abstract int length();
}

/**
 * Dictionary 클래스
 * {@code PairMap} 클래스 상속함
 *
 * Dictionary 클래스는 해시맵(혹은 딕셔너리) 자료형의 구현체입니다.
 * Dictionary 클래스는 특정 키 값에 대해 정보를 저장하고, 다시 쉽게 불러올 수 있도록 돕는 메서드들이 포함되어 있습니다.
 */
class Dictionary extends PairMap {
  /**
   * 딕셔너리가 담을 수 있는 자료 개수입니다.
```

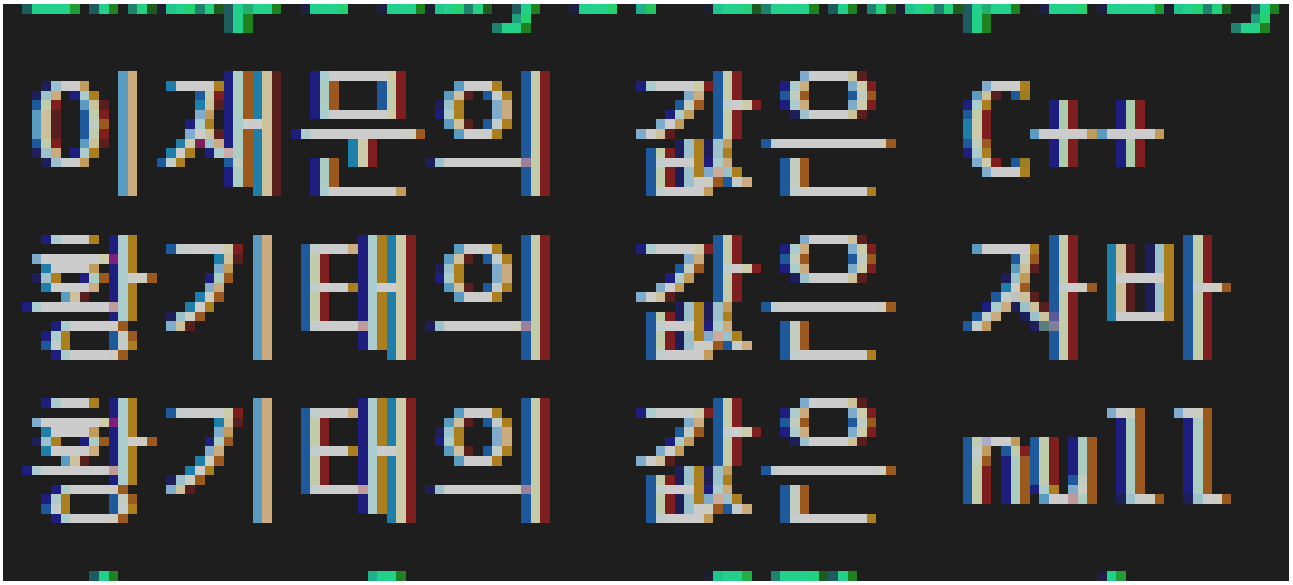
```
 */
int size;
/**
 * 딕셔너리에서 현재 유효한 값에 대한 정보입니다.
 * 만약 값이 사용 가능한 상태라면 {@code true}, 아니라면 {@code false}가 기록됩니다.
 */
boolean availables[];

/**
 * Dictionary 객체를 생성합니다.
 * @param size {@code int} 딕셔너리의 크기
 */
public Dictionary(int size) {
  this.size = size;
  // 크기에 맞춰 배열들 초기화
  this.keyArray = new String[size];
  this.valueArray = new String[size];
  this.availables = new boolean[size];
}

/**
 * {@code getter}
 * 딕셔너리에 기록된 값을 찾습니다. 만약 기록된 내용이 없다면 {@code null}를 반환합니다.
 * @param {@code key} 딕셔너리를 인덱싱할 인덱스 키
 * @return {@code String?} 딕셔너리의 인덱싱 결과
 */
@Override
String get(String key) {
  // 배열 크기만큼 반복문 시작
  for (int i = 0; i < this.size; i++) {
    // 만약 i번째 값이 유효한 값이 아니라면 다음 인덱스 확인
    // 유효하지 않은 값: 초기화되지 않음, 삭제됨
    if (!this.availables[i]) continue;
    // 만약 i번째 값이 유효한 값이라면
    // 함수 매개변수로 주어진 key와 keyArray의 현재 인덱싱 결과가 같은지 확인
    if (key.equals(this.keyArray[i])) {
      // 결과가 같다면 값(value) 반환
      return this.valueArray[i];
    }
  }
  // 이 지점까지 도달했다면 반복문을 모두 통과하고도
  // key값이 딕셔너리에 없다는 의미이므로 null 반환
  return null;
}

/**
 * {@code setter}
 * 딕셔너리에 값을 기록합니다.
 * 만약 딕셔너리가 꽉 차있다면 경고를 출력하고 함수가 종료됩니다.
 * 만약 매개 변수로 주어진 키로 딕셔너리를 인덱싱할 수 있다면 딕셔너리 내의 해당 값이 덮어쓰기됩니다.
 * @param key {@code String} 딕셔너리 키
 * @param value {@code String} 딕셔너리 값
 */
@Override
void put(String key, String value) {
  // 만약 get의 호출 결과가 null이라면 딕셔너리에 존재하지 않는 값이라는 의미임
  if (this.get(key) == null) {
    // 반복 시작: 딕셔너리에 사용 가능한 공간이 있는지 확인
    for (int i = 0; i < this.size; i++) {
      // 만약 현재 인덱스가 유효하지 않은 값을 가지고 있다면
      // 사용 가능한 공간임을 의미함
      if (!this.availables[i]) {
```

```
        this.availables[i] = true; // 현재 인덱스를 유효함으로 플래그
        // 딕셔너리 값 업데이트
        this.keyArray[i] = key;
        this.valueArray[i] = value;
        // 함수 종료
        return;
      }
    }
    // 오류 메시지 출력 후 함수 종료
    System.out.println("Error: No more spaces");
    return;
  } else {
    // 만약 get의 호출 결과가 null이 아니라면 이미 해당 키 값이 이 딕셔너리에 대해 유효하다는 의미임
    // 반복문 시작
    for (int i = 0; i < this.size; i++) {
      // 유효하지 않은 값을 찾는 것이 아니므로 유효하지 않은 값은 스킵
      if (!this.availables[i]) continue;
      // 만약 i번째 값이 유효한 값이라면
      // 함수 매개변수로 주어진 key와 keyArray의 현재 인덱싱 결과가 같은지 확인
      if (key.equals(this.keyArray[i])) {
        // 덮어쓰기 대상이라면 덮어쓰기 후 함수 종료
        this.valueArray[i] = value;
        return;
      }
    }
  }
}

/**
 * 딕셔너리에서 값을 제거합니다.
 * @param key {@code String} 제거할 값의 키
 * @return {@code String} 제거한 값
 */
@Override
String delete(String key) {
  // 반복문 시작
  for (int i = 0; i < this.size; i++) {
    // 만약 현재 인덱스 결과가 유효하지 않은 값이라면 다음 인덱스 처리
    if (!this.availables[i]) continue;
    // 만약 key값을 찾았다면
    if (key.equals(this.keyArray[i])) {
      // 실제로 삭제하지는 않음. 해당 공간이 비어있다고 표기한 후 나중에 덮어쓰기함.
      this.availables[i] = false;
      return this.valueArray[i]; // 삭제한 값 반환
    }
  }
  return null;
}

/**
 * 딕셔너리에 기록된 유효한 값들의 개수를 반환합니다.
 * @return {@code int} 유효한 값들의 개수
 */
@Override
int length() {
  // 카운터 선언
  int ables = 0;
  // 반복문 시작
  for (int i = 0; i < this.size; i++) {
    // 만약 현재 인덱스 결과가 유효하다면 카운터 1 증가
    if (this.availables[i]) ables++;
  }
```

```
    return ables;
  }
}
```

3.2 실행 예제

3.2.1 예제 1



4 과제 4

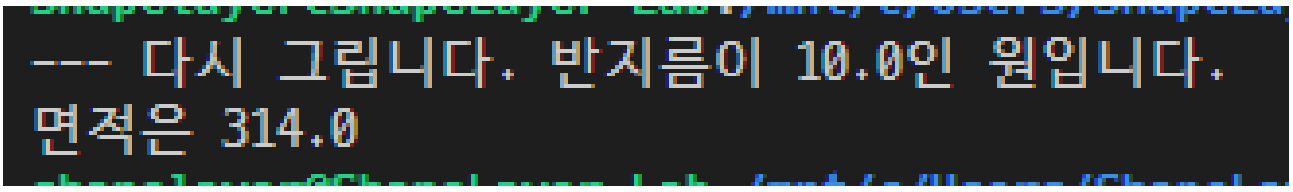
4.1 소스코드

```java
/**
 * @author @ShapeLayer
 */

// 문제 제시 코드
public class Main {
  public static void main(String [] args) {
    Shape donut = new Circle(10); // 반지름이 10인 원 객체
    donut.redraw();
    System.out.println("면적은 " + donut.getArea());
  }
}

// 문제 제시 인터페이스
interface Shape {
  final double PI = 3.14; // 상수
  void draw(); // 도형을 그리는 추상 메소드
  double getArea(); // 도형의 면적을 리턴하는 추상 메소드
  default public void redraw() { // 디폴트 메소드
    System.out.print("--- 다시 그립니다. ");
    draw();
  }
}

/**
 * Circle 클래스
 * {@code Shape} 인터페이스를 구현함
 *
 * Circle 클래스는 원을 그리고 처리하는데 유용한 메서드가 포함되어있습니다.
```

```
 */
class Circle implements Shape {
  /**
   * 원의 면적
   */
  double area;
  /**
   * 원의 반지름
   */
  double radius;

  /**
   * Circle 객체를 생성합니다.
   * @param radius {@code int} 원의 반지름
   */
  Circle(int radius) {
    this.radius = radius;
    this.area = this.radius * this.radius * this.PI;
  }

  /**
   * 원을 그리고 결과를 출력합니다.
   */
  @Override public void draw() {
    System.out.println("반지름이 " + this.radius + "인 원입니다.");
  }

  /**
   * 원의 면적을 반환합니다.
   * @return {@code double} 원의 면적
   */
  @Override public double getArea() { return this.area; }
}
```

4.2 실행 예제

4.2.1 예제 1



5 과제 5

5.1 소스코드

```
/**
 * @author @ShapeLayer
 */

// 문제 제시 코드
public class Main {
  static public void main(String [] args) {
    Shape [] list = new Shape[3]; // Shape을 상속받은 클래스 객체의 래퍼런스 배열
    list[0] = new Circle(10); // 반지름이 10인 원 객체
    list[1] = new Oval(20, 30); // 20x30 사각형에 내접하는 타원
    list[2] = new Rect(10, 40); // 10x40 크기의 사각형
    for(int i=0; i<list.length; i++) list[i].redraw();
```

```
    for(int i=0; i<list.length; i++) System.out.println("면적은 " + list[i].getArea());
  }
}

// 문제 제시 인터페이스
interface Shape {
  final double PI = 3.14; // 상수
  void draw(); // 도형을 그리는 추상 메소드
  double getArea(); // 도형의 면적을 리턴하는 추상 메소드
  default public void redraw() { // 디폴트 메소드
    System.out.print("--- 다시 그립니다. ");
    draw();
  }
}

/**
 * Circle 클래스
 * {@code Shape} 인터페이스를 구현함
 *
 * Circle 클래스는 원을 그리고 처리하는데 유용한 메서드가 포함되어있습니다.
 */
class Circle implements Shape {
  /**
   * 원의 면적
   */
  double area;
  /**
   * 원의 반지름
   */
  double radius;

  /**
   * Circle 객체를 생성합니다.
   * @param radius {@code int} 원의 반지름
   */
  Circle(int radius) {
    this.radius = radius;
    this.area = this.radius * this.radius * this.PI;
  }

  /**
   * 원을 그리고 결과를 출력합니다.
   */
  @Override public void draw() {
    System.out.println("반지름이 " + this.radius + "인 원입니다.");
  }

  /**
   * 원의 면적을 반환합니다.
   * @return {@code double} 원의 면적
   */
  @Override public double getArea() { return this.area; }
}

/**
 * Oval 클래스
 * {@code Shape} 인터페이스를 구현함
 *
 * Oval 클래스는 타원을 그리고 처리하는데 유용한 메서드가 포함되어 있습니다.
 */
class Oval implements Shape {
  /**
```

```
     * 타원의 면적
     */
    double area;
    /**
     * 타원의 가로 축
     */
    double width;
    /**
     * 타원의 세로 축
     */
    double height;

    /**
     * Oval 객체를 생성합니다.
     * @param width {@code int} 타원의 가로 축
     * @param height {@code int} 타원의 세로 축
     */
    Oval(int width, int height) {
      this.width = width;
      this.height = height;
      // 타원 면적 공식: PI*a*b
      this.area = this.PI * this.width * this.height;
    }

    /**
     * 타원을 그리고 결과를 출력합니다.
     */
    @Override public void draw() {
      System.out.println(String.format("%dx%d에 내접하는 타원입니다.", (int)width, (int)height));
    }
    /**
     * 타원의 면적을 반환합니다.
     * @return {@code double} 타원의 면적
     */
    @Override public double getArea() { return this.area; }
}

/**
 * Rect 클래스
 * {@code Shape} 인터페이스를 구현함
 *
 * Rect 클래스는 직사각형을 그리고 처리하는데 유용한 메서드가 포함되어 있습니다.
 */
class Rect implements Shape {
    /**
     * 직사각형의 면적
     */
    double area;
    /**
     * 직사각형의 가로 길이
     */
    double width;
    /**
     * 직사각형의 세로 길이
     */
    double height;

    /**
     * Rect 객체를 생성합니다.
     * @param width {@code int} 직사각형의 가로 길이
     * @param height {@code int} 직사각형의 세로 길이
     */
```

```
137     Rect(int width, int height) {
138       this.width = width;
139       this.height = height;
140       // 면적 공식: a*b
141       this.area = this.width * this.height;
142     }
143 
144     /**
145      * 직사각형을 그리고 결과를 출력합니다.
146      */
147     @Override public void draw() {
148       System.out.println(String.format("%dx%d크기의 사각형 입니다.", (int)width, (int)height));
149     }
150     /**
151      * 직사각형의 면적을 반환합니다.
152      * @return {@code double} 직사각형의 면적
153      */
154     @Override public double getArea() { return this.area; }
155 }
```

5.2 실행 예제

5.2.1 예제 1

```
--- 다시 그립니다. 반지름이 10.0인 원입니다.
--- 다시 그립니다. 20x30에 내접하는 타원입니다.
--- 다시 그립니다. 10x40크기의 사각형 입니다.
면적은 314.0
면적은 1884.0000000000002
면적은 400.0
```