12장 java.base 모듈

1. Object
2. System
3. 문자열
4. 포장
5. 수학
6. 날짜와 시간
7. 형식
8. 정규 표현식
9. 리플렉션
10. 어노테이션

1 Object Class

Object Class 예제

```
public class ObjectExample {
    public static void main(String[] args) {
        // 객체 생성
        String str = "Hello";

        // toString() 메서드를 호출하여 객체의 문자열 표현을 얻음
        String strRepresentation = str.toString();
        System.out.println("String representation: " + strRepresentation);

        // getClass() 메서드를 호출하여 객체의 클래스 정보를 얻음
        Class<?> classInfo = str.getClass();
        System.out.println("Class: " + classInfo.getName());

        // equals() 메서드를 사용하여 객체의 동등성을 확인
        String anotherStr = "Hello";
        boolean isEqual = str.equals(anotherStr);
        System.out.println("Equality: " + isEqual);
    }
}
```

Object 클래스는 모든 자바 클래스의 최상위 클래스이며, 자바 객체에 대한 일반적인 동작을 정합니다. 위의 예시 코드에서는 Object 클래스의 주요 메서드인 **toString()**, **getClass()**, **equals()**를 사용하여 객체의 문자열 표현, 클래스 정보, 그리고 객체 간의 동등성을 확인하는 방법을 보여줍니다.

Object Class

Object Class 예제 - 리팩토링

```
public class ObjectExample {
    public static void main(String[] args) {
        String str = "Hello";

        printStringRepresentation(str);
        printClassInformation(str);
        checkEquality(str, "Hello");
    }

    private static void printStringRepresentation(Object obj) {
        String strRepresentation = obj.toString();
        System.out.println("String representation: " + strRepresentation);
    }

    private static void printClassInformation(Object obj) {
        Class<?> classInfo = obj.getClass();
        System.out.println("Class: " + classInfo.getName());
    }

    private static void checkEquality(Object obj1, Object obj2) {
        boolean isEqual = obj1.equals(obj2);
        System.out.println("Equality: " + isEqual);
    }
}
```

코드의 중복을 제거하고 각각의 작업을 메서드로 분리함으로써 코드의 재사용성을 높이고, 유지보수성을 향상시킬 수 있음

1.toString() 메서드 호출과 클래스 정보를 출력하는 코드를 각각의 메서드로 분리하여 가독성을 향상시켰습니다.
2.객체의 동등성을 확인하는 부분도 별도의 메서드로 분리하여 코드의 중복을 제거하고 재사용성을 높였습니다.
3.메인 메서드에서는 분리된 메서드들을 호출하여 각각의 작업을 수행하도록 변경하여 메인 메서드의 가독성을 높였습니다.

2 System Class

Sytem Class 예제

```
public class SystemExample {
    public static void main(String[] args) {
        // 콘솔에 메시지 출력
        System.out.println("Hello, World!");

        // 현재 시간(milliseconds)을 얻기
        long currentTime = System.currentTimeMillis();
        System.out.println("Current time in milliseconds: " + currentTime);

        // 시스템 종료
        System.exit(0);
    }
}
```

System 클래스는 자바 프로그램과 시스템 간의 상호작용을 제공하는 클래스입니다. 위의 예시 코드에서는 콘솔에 메시지 출력, 현재 시간(milliseconds)을 얻는 방법, 그리고 시스템 종료 방법을 보여줍니다.

2 System Class

Sytem Class 예제 - 리팩토링

```
import java.time.Instant;

public class SystemExample {
    private static final boolean EXIT_PROGRAM = false;

    public static void main(String[] args) {
        // 콘솔에 메시지 출력
        printMessage("Hello, World!");

        // 현재 시간(milliseconds)을 얻기
        printCurrentTime();

        // 시스템 종료 여부 확인 및 종료
        if (EXIT_PROGRAM) {
            exitProgram();
        }
    }

    private static void printMessage(String message) {
        System.out.println(message);
    }

    private static void printCurrentTime() {
        long currentTime = Instant.now().toEpochMilli();
        System.out.println("Current time in milliseconds: " +
currentTime);
    }

    private static void exitProgram() {
        System.exit(0);
    }
}
```

1.System.out.println() 호출을 별도의 메서드로 분리하여 코드의 가독성을 향상시켰습니다.
2.현재 시간을 가져오는데 System.currentTimeMillis() 대신에 Java 8에서 추가된 java.time.Instant.now().toEpochMilli()를 사용하여 가독성을 높였습니다.

3 문자열 Class

문자열 Class 예제

```
public class StringExample {
    public static void main(String[] args) {
        // 문자열 생성
        String str1 = "Hello";
        String str2 = "World";

        // 문자열 연결(concatenation)
        String result = str1 + ", " + str2 + "!";
        System.out.println(result);

        // 문자열의 길이(length)
        int length = result.length();
        System.out.println("Length: " + length);

        // 특정 문자 또는 부분 문자열 검색(indexOf)
        int index = result.indexOf("World");
        System.out.println("Index of 'World': " + index);

        // 문자열 분할(split)
        String[] parts = result.split(",");
        System.out.println("Split parts:");
        for (String part : parts) {
            System.out.println(part.trim());
        }

        // 대소문자 변환(toLowerCase, toUpperCase)
        String upperCase = result.toUpperCase();
        String lowerCase = result.toLowerCase();
        System.out.println("Uppercase: " + upperCase);
        System.out.println("Lowercase: " + lowerCase);
    }
}
```

String 클래스는 자바에서 문자열을 나타내는 데 사용되는 클래스입니다. 위의 예시 코드에서는 문자열의 연결(concatenation), 길이(length), 특정 문자 또는 부분 문자열 검색(indexOf), 문자열 분할(split), 대소문자 변환(toLowerCase, toUpperCase) 등의 기본적인 문자열 작업을 보여줍니다.

3 문자열 Class

문자열 Class 예제 - 리팩토링

```
public class StringExample {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "World";

        // 문자열 연결(concatenation)
        String result = concatenateStrings(str1, str2);
        System.out.println(result);

        // 문자열의 길이(length)
        printStringLength(result);

        // 특정 문자 또는 부분 문자열 검색(indexOf)
        printIndexOf(result, "World");

        // 문자열 분할(split)
        printSplitParts(result);

        // 대소문자 변환(toLowerCase, toUpperCase)
        printCaseConversions(result);
    }

    private static String concatenateStrings(String str1, String str2) {
        return str1 + ", " + str2 + "!";
    }

    private static void printStringLength(String str) {
        System.out.println("Length: " + str.length());
    }

    private static void printIndexOf(String str, String target) {
        System.out.println("Index of '" + target + "': " +
str.indexOf(target));
    }

    private static void printSplitParts(String str) {
        String[] parts = str.split(",");
        System.out.println("Split parts:");
        for (String part : parts) {
            System.out.println(part.trim());
        }
    }

    private static void printCaseConversions(String str) {
        System.out.println("Uppercase: " + str.toUpperCase());
        System.out.println("Lowercase: " + str.toLowerCase());
    }
}
```

4 포장 Class

포장 Class 예제

```
public class WrapperExample {
    public static void main(String[] args) {
        // 기본 자료형의 값을 포장 객체로 변환
        Integer intValue = Integer.valueOf(10);
        Double doubleValue = Double.valueOf(3.14);

        // 포장 객체에서 기본 자료형 값 얻기
        int intVal = intValue.intValue();
        double doubleVal = doubleValue.doubleValue();
        System.out.println("Integer value: " + intVal);
        System.out.println("Double value: " + doubleVal);

        // 문자열을 포장 객체로 변환
        Integer stringToInt = Integer.parseInt("100");
        System.out.println("Parsed integer value: " +
stringToInt);

        // 포장 객체에서 문자열 얻기
        String intToString = stringToInt.toString();
        System.out.println("Integer as string: " + intToString);
    }
}
```

포장 클래스는 기본 자료형을 객체로 래핑하는 데 사용됩니다. 위의 예시 코드에서는 기본 자료형 값과 문자열 간의 변환, 그리고 포장된 값에서 기본 자료형 값으로의 변환 등을 보여줍니다.

포장 Class

포장 Class 예제 - 리팩토링

```
public class WrapperExample {
    public static void main(String[] args) {
        Integer intValue = wrapPrimitive(10);
        Double doubleValue = wrapPrimitive(3.14);

        // 포장 객체에서 기본 자료형 값 얻기
        unwrapPrimitive(intValue, doubleValue);

        // 문자열을 포장 객체로 변환
        Integer stringToInt = parseAndWrap("100");
        printParsedInteger(stringToInt);

        // 포장 객체에서 문자열 얻기
        String intToString = unwrapAndConvert(intValue);
        printIntegerAsString(intToString);
    }

    private static <T> T wrapPrimitive(T value) {
        return value;
    }

    private static void unwrapPrimitive(Integer intValue,
Double doubleValue) {
        System.out.println("Integer value: " + intValue);
        System.out.println("Double value: " + doubleValue);
    }

    private static Integer parseAndWrap(String value) {
        return Integer.parseInt(value);
    }

    private static void printParsedInteger(Integer value) {
        System.out.println("Parsed integer value: " + value);
    }

    private static String unwrapAndConvert(Integer intValue)
{
        return intValue.toString();
    }

    private static void printIntegerAsString(String value) {
        System.out.println("Integer as string: " + value);
    }
}
```

•기본 자료형을 포장하는 부분과 포장 객체를 기본 자료형으로 얻는 부분을 각각의 메서드로 분리하여 코드를 모듈화 하였습니다.
•이렇게 하면 코드의 가독성을 높이고, 메서드의 역할을 명확히 할 수 있습니다.

5 수학 Class

수학 Class 예제

```
public class MathExample {
    public static void main(String[] args) {
        // 절대값 구하기
        double absolute = Math.abs(-10.5);
        System.out.println("Absolute value: " + absolute);

        // 제곱근 구하기
        double squareRoot = Math.sqrt(25);
        System.out.println("Square root: " + squareRoot);

        // 두 값 중 최댓값 구하기
        int max = Math.max(10, 20);
        System.out.println("Maximum value: " + max);

        // 두 값 중 최솟값 구하기
        int min = Math.min(30, 40);
        System.out.println("Minimum value: " + min);

        // 랜덤한 값 생성
        double random = Math.random();
        System.out.println("Random value: " + random);
    }
}
```

Math 클래스는 수학적인 연산을 제공하는 클래스입니다. 위의 예시 코드에서는 절대값, 제곱근, 최댓값, 최솟값을 구하는 방법 및 랜덤한 값 생성 방법 등을 보여줍니다.

5 수학 Class

수학 Class 예제 - 리팩토링

```
public class MathExample {
    public static void main(String[] args) {
        printAbsoluteValue(-10.5);
        printSquareRoot(25);
        printMaxValue(10, 20);
        printMinValue(30, 40);
        printRandomValue();
    }

    private static void printAbsoluteValue(double number) {
        System.out.println("Absolute value: " +
Math.abs(number));
    }

    private static void printSquareRoot(double number) {
        System.out.println("Square root: " +
Math.sqrt(number));
    }

    private static void printMaxValue(int x, int y) {
        System.out.println("Maximum value: " + Math.max(x,
y));
    }

    private static void printMinValue(int x, int y) {
        System.out.println("Minimum value: " + Math.min(x,
y));
    }

    private static void printRandomValue() {
        System.out.println("Random value: " +
Math.random());
    }
}
```

이제 각각의 메서드는 특정 작업을 수행하고, 메인 메서드에서는 이러한 메서드를 호출하여 코드를 간결하고 가독성 있게 유지할 수 있습니다.

6 날짜와 시간 Class

날짜와 시간 Class 예제

```
import java.util.Date;

public class DateTimeExample {
    public static void main(String[] args) {
        // 현재 날짜와 시간 얻기
        Date currentDate = new Date();
        System.out.println("Current date and time: " +
currentDate);

        // 특정 시간으로 날짜 객체 생성
        Date specificDate = new Date(121, 2, 13); // 2021년
3월 13일
        System.out.println("Specific date: " + specificDate);
    }
}
```

java.util 패키지의 Date 클래스는 날짜와 시간 정보를 나타내는 데 사용됩니다. 위의 예시 코드에서는 현재 날짜와 시간을 얻는 방법 및 특정 시간으로 날짜 객체를 생성하는 방법을 보여줍니다.

형식 Class

형식 Class 예제

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class FormattingExample {
    public static void main(String[] args) {
        // 날짜 형식 지정
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");

        // 현재 날짜와 시간을 지정된 형식으로 출력
        String formattedDate = sdf.format(new Date());
        System.out.println("Formatted date: " + formattedDate);
    }
}
```

형식 클래스는 날짜와 시간을 지정된 형식에 맞게 출력하는 데 사용됩니다. 위의 예시 코드에서는 SimpleDateFormat 클래스를 사용하여 날짜를 지정된 형식으로 출력하는 방법을 보여줍니다.

8 정규 표현식 Class

정규 표현식 Class 예제

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexExample {
    public static void main(String[] args) {
        // 정규 표현식 패턴 생성
        Pattern pattern = Pattern.compile("₩₩d+"); // 숫자 패턴

        // 정규 표현식 매칭
        Matcher matcher = pattern.matcher("Hello123World456");

        // 매칭된 결과 출력
        while (matcher.find()) {
            System.out.println("Match: " + matcher.group());
        }
    }
}
```

정규 표현식은 문자열에서 패턴을 찾거나 매칭하기 위해 사용됩니다. java.util.regex 패키지의 Pattern 및 Matcher 클래스를 사용하여 정규 표현식을 구현할 수 있습니다. 위의 예시 코드에서는 숫자 패턴에 해당하는 부분 문자열을 찾아 출력하는 방법을 보여줍니다.

9 리플렉션 Class

리플렉션 Class 예제

```
import java.lang.reflect.Method;

public class ReflectionExample {
    public static void main(String[] args) throws Exception {
        // 클래스 이름으로 클래스 객체 얻기
        Class<?> cls = Class.forName("java.lang.String");

        // 클래스의 메서드 정보 얻기
        Method[] methods = cls.getMethods();

        // 메서드 정보 출력
        for (Method method : methods) {
            System.out.println("Method name: " + method.getName());
        }
    }
}
```

리플렉션은 실행 중인 프로그램 내부에서 클래스의 정보를 분석하고 조작하는 데 사용됩니다. java.lang.reflect 패키지의 클래스들을 사용하여 클래스의 메서드, 필드 등의 정보를 얻을 수 있습니다. 위의 예시 코드에서는 String 클래스의 메서드 정보를 얻고 출력하는 방법을 보여줍니다.

10 어노테이션 Class

어노테이션 Class 예제

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface MyAnnotation {
    String value();
}

public class AnnotationExample {
    @MyAnnotation("Hello")
    public static void myMethod() {
        System.out.println("Inside annotated method");
    }

    public static void main(String[] args) throws Exception {
        // 메서드에 부여된 어노테이션 정보 가져오기
        Method method =
AnnotationExample.class.getMethod("myMethod");
        MyAnnotation annotation =
method.getAnnotation(MyAnnotation.class);
        // 어노테이션 값 출력
        System.out.println("Annotation value: " + annotation.value());
    }
}
```

어노테이션은 코드에 메타데이터를 부여하여 컴파일러에게 정보를 전달하거나 실행 중에 리플렉션을 통해 정보를 분석할 수 있습니다. 위의 예시 코드에서는 @MyAnnotation이라는 사용자 정의 어노테이션을 정의하고 이를 메서드에 적용하여 어노테이션 값을 출력하는 방법을 보여줍니다.

감사합니다