확인문제

Chapter 11

1. 예외에 대한 설명 중 틀린 것은 무엇입니까?

❶ 예외는 사용자의 잘못된 조작, 개발자의 잘못된 코딩으로 인한 프로그램 오류를 말한다.

❷ RuntimeException의 하위 예외는 컴파일러가 예외 처리 코드를 체크하지 않는다.

❸ 예외는 try-catch 블록을 사용해서 처리된다.

❹ 자바 표준 예외만 프로그램에서 처리할 수 있다.

2. try-catch-finally 블록에 대한 설명 중 틀린 것은 무엇입니까?

❶ try {} 블록에는 예외가 발생할 수 있는 코드를 작성한다.

❷ catch {} 블록은 try {} 블록에서 발생한 예외를 처리하는 블록이다.

❸ try {} 블록에서 return 문을 사용하면 finally {} 블록은 실행되지 않는다.

❹ catch {} 블록은 예외의 종류별로 여러 개를 작성할 수 있다.

3. throws에 대한 설명으로 틀린 것은 무엇입니까?

❶ 생성자나 메소드의 선언 끝 부분에 사용되어 내부에서 발생된 예외를 떠넘긴다.

❷ throws 뒤에는 떠넘겨야 할 예외를 쉼표(,)로 구분해서 기술한다.

❸ 모든 예외를 떠넘기기 위해 간단하게 throws Exception으로 작성할 수 있다.

❹ 새로운 예외를 발생시키기 위해 사용된다.

4. throw에 대한 설명으로 틀린 것은 무엇입니까?

❶ 예외를 최초로 발생시키는 코드이다.

❷ 예외를 호출한 곳으로 떠넘기기 위해 메소드 선언부에 작성된다.

❸ throw로 발생된 예외는 일반적으로 생성자나 메소드 선언부에 throws로 떠넘겨진다.

❹ throw 키워드 뒤에는 예외 객체 생성 코드가 온다.

5. 메소드가 다음과 같이 선언되어 있습니다. 잘못된 예외 처리를 선택하세요.

```
public void method1() throws NumberFormatException, ClassNotFoundException { … }
```

❶ try { method1(); } catch (Exception e) {}

❷ void method2() throws Exception { method1(); }

❸ try { method1(); }
catch (Exception e) {}
catch (ClassNotFoundException e) {}

❹ try { method1(); }
catch (ClassNotFoundException e) {}
catch (NumberFormatException e) {}

6. 다음 코드가 실행되었을 때 출력 결과를 작성해보세요.

```
String[] strArray = { "10", "2a" };
int value = 0;
for(int i=0; i<=2; i++) {
  try {
    value = Integer.parseInt(strArray[i]);
  } catch(ArrayIndexOutOfBoundsException e) {
    System.out.println("인덱스를 초과했음");
  } catch(NumberFormatException e) {
    System.out.println("숫자로 변환할 수 없음");
  } finally {
    System.out.println(value);
  }
}
```

확인문제

7. login() 메소드에서 존재하지 않는 ID를 입력하면 NotExistIDException을 발생시키고, 잘못된 패스워드를 입력하면 WrongPasswordException을 발생시키려고 합니다. 다음 LoginExample의 실행 결과를 보고 빈칸을 채워보세요.

```
public class NotExistIDException extends Exception {
  public NotExistIDException() {}
  public NotExistIDException(String message) {
    
  }
}
```

```
public class WrongPasswordException extends Exception {
  public WrongPasswordException() {}
  public WrongPasswordException(String message) {
    
  }
}
```

```
public class LoginExample {
  public static void main(String[] args) {
    try {
      login("white", "12345");
    } catch(Exception e) {
      System.out.println(e.getMessage());
    }

    try {
      login("blue", "54321");
    } catch(Exception e) {
      System.out.println(e.getMessage());
    }
  }

  public static void login(String id, String password)  {
    //id가 blue가 아니라면 NotExistIDException을 발생시킴
    if(!id.equals("blue")) {
      
    }
```

```
        //password가 12345가 아니라면 WrongPasswordException을 발생시킴
        if(!password.equals("12345")) {
            
        }
    }
}
```

실행 결과

```
아이디가 존재하지 않습니다.
```

실행 결과

```
패스워드가 틀립니다.
```

8. FileWriter는 파일을 열고 데이터를 저장하는 클래스입니다. 예외 발생 여부와 상관 없이 마지막에는 close() 메소드를 실행해서 파일을 닫아주려고 합니다. 왼쪽 코드는 try-catch-finally를 이용해서 작성한 코드로, 리소스 자동 닫기를 이용하도록 수정하고 싶습니다. 수정한 코드를 오른쪽에 작성해보세요.

```
import java.io.IOException;

public class FileWriter implements AutoCloseable {
  public FileWriter(String filePath) throws IOException {
    System.out.println(filePath + " 파일을 엽니다.");
  }

  public void write(String data) throws IOException {
    System.out.println(data + "를 파일에 저장합니다.");
  }

  @Override
  public void close() throws IOException {
    System.out.println("파일을 닫습니다.");
  }
}
```

확인문제

```
import java.io.IOException;

public class FileWriterExample {
  public static void main(String[]
      args) {
    FileWriter fw = null;
    try {
      fw = new FileWriter("file.txt");
      fw.write("Java");
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      try { fw.close(); } catch
        (IOException e) {}
    }
  }
}
```

```
import java.io.IOException;

public class FileWriterExample {
  public static void main(String[]
      args) {
    [                    ]
  }
}
```