**실습 제출물**

1. **First Task에서 탐지한 오류 (1점)**

import java.util.ArrayList;

public class User{

private int id;

private String firstName;

private String lastName;

private int age;

private String profession;

private ArrayList<User> children;

public User(int id,String firstName,String lastName,int age){

this.id = id;

this.firstName = firstName;

this.lastName = lastName;

this.age = age;

this.profession = profession;

}

public String getChildName(Userchild){

if(!this.children.contains(child)){

new Exception("Invalidargument!");

}

else{

String name = null;

if(child.getFirstName()!=null){

name = child.getFirstName();

}

if(name =="Harry"){

name.replace('r','p');

}

if(name != null || name.length() > 0){

name.concat(child.getLastName());

}

}

return this.getChildName(child);

}

public String getFirstName(){

return firstName;

}

public String getLastName(){

return lastName;

}

}

1. **HospitalSystem 프로젝트에서 SpotBugs 도구로 발견한 버그 10개를 분석하시오. 모든 다른 등급 (Of Concern, Troubling, Scary, Scariest) 및 confidences(Low, Normal, High)에서 버그를 선택하고 주어진 버그가 실제 오류인지 거짓 양성(오류가 아닌데 오류라고 함)인지 적으세요. 각 버그는 다음 표에 기입하라. (7점)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 버그 설명 및 위치 | 문제 설명 | 실제 오류인가, 가짜 오류인가?  버그를 수정해야 하는가?  어떻게 수정? 아니라면 왜? |
| 0 | Exception created and dropped rather than thrown  Person.java:15 | This code creates an exception (or error) object, but doesn't do anything with it. | This is a true positive and should be fixed. Besides simply creating the instance of the exception, it should also be thrown.  The fixed line would be:  *throw new Exception("Birthyear cannot be in the future!");* |
| 1 | Method ignores return value  Person.java:29 | The return value of this method should be checked. One common cause of this warning is to invoke a method on an immutable object, thinking that it updates the object. | This is a true positive and should be fixed. After The toLowerCase() function returns new String Value, it should also be saved at Original variable.  The fixed line would be:  firstName = firstName.toLowerCase(); |
| 2 | Bad constant value for month  Hospital.java:41 | This code passes a constant month value outside the expected range of 0..11 to a method. | This is a true positive and should be fixed. This is because the range of months is 1 to 12. Therefore, no error should occur even when 12 is entered.  The fixed line would be:  doctors.add(**new** Doctor("Hermione", "Granger", **new** GregorianCalendar(1988, 12 - 1, 10), Specialty.***PSY***, room102)); |
| 3 | Covariant equals() method defined, Object.equals(Object) inherited  ExaminationRoom.java:25 | This class defines a covariant version of the equals() method, but inherits the normal equals(Object) method defined in the base java.lang.Object class.  The class should probably define a boolean equals(Object) method. | This is a false positive and should not be fixed. Because that's just method overwriting that overrides the equals() method. |
| 4 | Possible null pointer dereference.  Hospital.java:102 | There is a branch of statement that, if executed, guarantees that a null value will be dereferenced, which would generate a NullPointerException when the code is executed. Of course, the problem might be that the branch or statement is infeasible and that the null pointer exception cannot ever be executed; deciding that is beyond the ability of SpotBugs. | This is a false positive and should not be fixed. Since the examinationRoom object is initialized inside the if statement, NullPointerException does not occur. |
| 5 | Random object created and used only once  Patient.java:17 | This code creates a java.util.Random object, uses it to generate one random number, and then discards the Random object. This produces mediocre quality random numbers and is inefficient. If possible, rewrite the code so that the Random object is created once and saved, and each time a new random number is required invoke a method on the existing Random object to obtain it.  If it is important that the generated Random numbers not be guessable, you *must* not create a new Random for each random number; the values are too easily guessable. You should strongly consider using a java.security.SecureRandom instead (and avoid allocating a new SecureRandom for each random number needed). | This is a false positive and should not be fixed. There is no problem because the Random object is used only once and does not generate an error. |
| 6 | Comparison of String objects using == or !=  Hospital.java:66 | This code compares java.lang.String objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the String.intern() method, the same string value may be represented by two different String objects. Consider using the equals(Object) method instead. | This is a true positive and should be fixed. This is because == or != operators cannot check whether the strings held by the object are the same.  The fixed line would be:  **if** (thisNumber.equals(roomNumber)) |
| 7 | Write to static field from instance method  Hospital.java:21 | This instance method writes to a static field. This is tricky to get correct if multiple instances are being manipulated, and generally bad practice. | This is a false positive and should not be fixed. This is because the array is assigned to a static variable and no error occurs. |
| 8 | Use the niextInt method of Random rather than nextDouble to generate a random integer  Patient.java:17 | If r is a java.util.Random, you can generate a random number from 0 to n-1 using r.nextInt(n), rather than using (int)(r.nextDouble() \* n).  The argument to nextInt must be positive. If, for example, you want to generate a random value from -99 to 0, use -r.nextInt(100). | This is a false positive and should not be fixed. This is because conversion from double to int causes information loss but no error. |
| 9 | Method invokes inefficient new String() constructor  UI.java:327 | Creating a new java.lang.String object using the no-argument constructor wastes memory because the object so created will be functionally indistinguishable from the empty string constant "".  Java guarantees that identical string constants will be represented by the same String object.  Therefore, you should just use the empty string constant directly. | This is a false positive and should not be fixed. This is because constructing a new java.lang.String object using the no-arg constructor is functionally indistinguishable from the empty string constant "", so memory is wasted and no error is raised. |
| 10 | Consider using Locale parameterized version of invoked method  Person.java:29 | A String is being converted to upper or lowercase, using the platform's default encoding. This may result in improper conversions when used with international characters | This is a false positive and should not be fixed. This is because only a message to convert to the language explicitly set by setting the Locale is displayed, and the conversion is based on English by default, and no error occurs. |

1. **방법 비교 (2점)**

정적 코드 분석 도구와 동적 테스팅의 이점 간략하게 비교. 두가지 방법 모두 최소 한 가지의 구체적인 이점을 설명하시오.

정적 코드 분석 도구의 이점:

1. 버그와 보안 취약점 식별: 정적 코드 분석 도구는 소스 코드를 분석하여 잠재적인 버그나 보안 취약점을 식별할 수 있습니다. 이를 통해 개발자는 이러한 문제를 빠르게 파악하고 수정할 수 있습니다.
2. 비용 효율성: 정적 코드 분석 도구는 자동화된 방식으로 소스 코드를 분석하기 때문에 인력 및 시간을 절약할 수 있습니다. 개발자들은 수동으로 코드를 검토하는 대신 도구를 사용하여 문제를 찾을 수 있습니다.

동적 테스팅의 이점:

1. 실제 환경 시뮬레이션: 동적 테스팅은 소프트웨어를 실행하여 실제 환경에서의 동작을 시뮬레이션합니다. 이를 통해 소프트웨어의 동작을 직접 확인하고 버그나 예외 상황을 식별할 수 있습니다.
2. 런타임 문제 검출: 동적 테스팅은 소프트웨어를 실행하면서 런타임에 발생하는 문제를 검출할 수 있습니다. 예를 들어, 메모리 누수나 성능 저하와 같은 문제를 실시간으로 감지할 수 있습니다.