

# BÀI 6. CÁC QUY TẮC LẬP TRÌNH AN TOÀN TRONG JAVA

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

1

1

## 1. GIỚI THIỆU CHUNG

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

2

2

## ATBM trong chương trình Java

- Về cơ bản, Java là một ngôn ngữ khá an toàn:
  - Không có toán tử trên con trả
  - Tự động kiểm tra truy cập vượt ngoài xâu, mảng
  - Tự động kiểm tra con trả null
  - Tự động kiểm tra tương thích về kiểu dữ liệu
  - Các toán tử và kiểu dữ liệu không phụ thuộc vào hệ điều hành và phần cứng
- Tuy nhiên, vẫn còn nhiều vấn đề cần phải kiểm soát an toàn trong chương trình Java.

3

3

## ATBM trong chương trình Java

- Chương trình phần mềm có nhiều thành phần khác nhau:
  - Mỗi thành phần có mức độ tin cậy khác nhau
  - Mỗi thành phần xử lý trong những miền tin cậy khác nhau
  - Một số thành phần là đáng tin cậy(Trusted Computing Base), nhưng phần lớn còn lại thì không
  - Chương trình có những thành phần do bên thứ 3 phát triển
    - tồn tại ranh giới tin cậy (trust boundary) giữa các thành phần
- Nguyên tắc chung: dữ liệu phải được kiểm duyệt khi đi qua ranh giới tin cậy

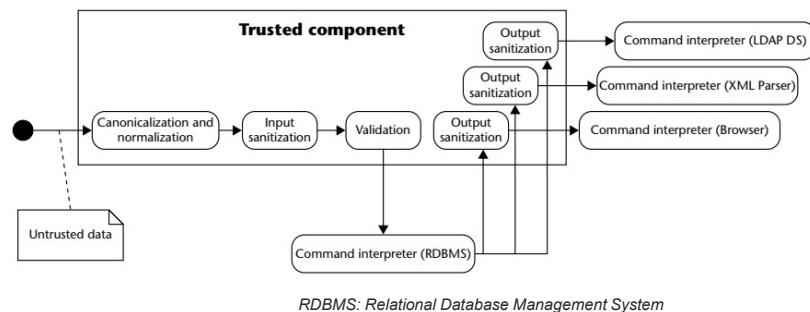
4

4

## ATBM trong chương trình Java

- Tấn công tiêm nhiễm (Injection Attack):

- Dữ liệu đi từ ngoài ranh giới bảo mật vào bên trong miền được tin cậy có thể là độc hại
- Dữ liệu đi ra có thể là độc hại khi một thành phần trong miền bị đầu độc



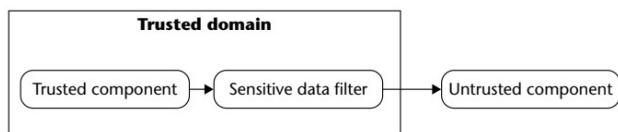
5

5

## ATBM trong chương trình Java

- Lộ lọt dữ liệu nhạy cảm: dữ liệu nhạy cảm được chuyển từ thành phần tin cậy sang thành phần không đáng tin

- Cần xác định mức độ nhạy cảm của các dữ liệu
- Dữ liệu nhạy cảm cần phải được lọc khi đi qua ranh giới tin cậy



- Kiểm soát năng lực truy cập: khi một lớp có các dữ liệu và phương thức nhạy cảm thì tham chiếu của đối tượng đó cũng phải là dữ liệu nhạy cảm và không được lộ lọt tới miền không đáng tin cậy

6

6

## ATBM trong chương trình Java

- Tấn công từ chối dịch vụ: khiến cho tài nguyên không sẵn sàng hoặc bị thiết hụt khi có yêu cầu truy cập
  - Làm cạn kiệt tài nguyên: RAM, CPU, ổ cứng, băng thông, định danh
  - Lợi dụng lỗ hổng không xử lý ngoại lệ
  - Lợi dụng lỗ hổng trong mô hình đa luồng, ví dụ: deadlock
- Kiểm soát năng lực truy cập: khi một lớp có các dữ liệu và phương thức nhạy cảm thì tham chiếu của đối tượng đó cũng phải là dữ liệu nhạy cảm và không được lộ lọt tới miền không đáng tin cậy

7

7

## Lập trình an toàn là gì?

- Yêu cầu: Viết mã nguồn chương trình để đạt được các mục tiêu an toàn bảo mật
- Bao gồm nhiều kỹ thuật khác nhau:
  - Kiểm soát giá trị đầu vào
  - Kiểm soát truy cập bộ nhớ chính
  - Che giấu mã nguồn
  - Chống dịch ngược
  - Chống giả mạo mã nguồn
  - Kiểm soát kết quả đầu ra
  - Kiểm soát quyền truy cập
  - ...

8

8

## Các mục tiêu an toàn bảo mật

- Confidentiality (Bí mật): thông tin chỉ có thể xem bởi chủ thể được cấp quyền
- Integrity (Toàn vẹn): thông tin chỉ có thể chỉnh sửa bởi chủ thể được cấp quyền
- Availability (Sẵn sàng): hệ thống sẵn sàng đáp ứng khi có yêu cầu
  - Dễ dàng truy cập và sử dụng
  - Thời gian đáp ứng
  - Khả năng chịu lỗi
  - Khả năng phục hồi



9

9

## SEI CERT Oracle Secure Coding for Java

- Được giới thiệu chuẩn hóa bởi đại học Carnegie Mellon
- Mỗi quy tắc được đánh giá theo 3 tiêu chí:
  - Severity: Mức độ nghiêm trọng nếu không được tuân thủ
    - ✓ Low: tấn công DoS, kết thúc bất thường
    - ✓ Medium: vi phạm toàn vẹn dữ liệu, lọt dữ liệu
    - ✓ High: thực thi mã độc, leo thang đặc quyền
  - Likelihood: Khả năng có thể bị khai thác nếu không được tuân thủ
    - ✓ Unlikely: Không chắc
    - ✓ Probable: Có thể
    - ✓ Likely: rất có thể
  - Cost: Chi phí để vá lỗi
    - ✓ High: Phát hiện và sửa lỗi thủ công
    - ✓ Medium: Phát hiện tự động và sửa thủ công
    - ✓ Low: Phát hiện và sửa lỗi tự động
- Priority = Severity \* Likelihood \* Cost

Level 1: P12 – P27  
Level 2: P6 – P9  
Level 3: P1 – P4

10

10

## 2. KIỂM DUYỆT ĐẦU VÀO VÀ LÀM SẠCH DỮ LIỆU

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

11

11

### Quy tắc IDS00-J

- Làm sạch tất cả dữ liệu đầu vào không tin cậy  
được gửi qua ranh giới tin cậy
- Một số loại dữ liệu đầu vào có thể:
  - Các giá trị do người dùng cung cấp
  - File được mở
  - Các gói tin nhận được từ mạng
  - Các dữ liệu thu nhận từ thiết bị cảm biến (Ví dụ: QR code, âm thanh, hình ảnh,...)
  - Thư viện của bên thứ 3
- Không tuân thủ quy tắc này sẽ dẫn đến hàng loạt lỗ hổng nghiêm trọng

12

12

## Ví dụ: SQL Injection

```
public void doPrivilegedAction( String username, char[] password)
throws SQLException {
    Connection connection = getConnection();
    if (connection == null) { // handle error
        try {
            String pwd = hashPassword(password);
            String sqlString = "SELECT * FROM db_user WHERE username =
                '" + username + "' AND password = '" + pwd + "'";
            Statement stmt = connection.createStatement();
            ResultSet rs = stmt.executeQuery(sqlString);
            if (!rs.next()) {
                throw new SecurityException("User name or password incorrect");
            }
            // Authenticated; proceed
        } finally {
            try {
                connection.close();
            } catch (SQLException x) {
                // forward to handler
            }
        }
    }
}
```

13

13

## Ví dụ: SQL Injection – Sửa lỗi

```
// Ensure that the length of user name is legitimate
if ((password.length() < 8) {
// Handle error
}
String pwd = hashPassword(password);
String sqlString = "select * from db_user where username=? And
                    password=?";
PreparedStatement stmt = connection.prepareStatement(sqlString);
stmt.setString(1, username);
stmt.setString(2, pwd);
ResultSet rs = stmt.executeQuery();
if (!rs.next()) {
throw new SecurityException("User name or password incorrect");
}
```

14

14

## Ví dụ: XML Injection

- Thông tin giở hàng được đóng gói vào cấu trúc XML

```
<item>
    <description>Widget</description>
    <price>500.0</price>
    <quantity>1</quantity>
</item>
```

- Mã nguồn có lỗi

```
private void createXMLStream(BufferedOutputStream outStream,
    String quantity) throws IOException {
    String xmlString;
    xmlString = "<item>\n<description>Widget</description>\n" +
        "<price>500.0</price>\n" +
        "<quantity>" + quantity + "</quantity></item>";
    outStream.write(xmlString.getBytes());
    outStream.flush();
}
```

15

15

## Ví dụ: XML Injection

- Nếu trường quantity được truyền vào giá trị sau:

```
1</quantity><price>1.0</price><quantity>1
```

- Cấu trúc XML trở thành

```
<item>
    <description>Widget</description>
    <price>500.0</price>
    <quantity>1</quantity><price>1.0</price><quantity>1</quantity>
</item>
```

- Khi xử lý bằng SAX Parser trong Java thì giá trị của trường price thứ hai sẽ đè lên giá trị của trường đầu

16

16

## Ví dụ: XML Injection – Sửa lỗi

- Sử dụng danh sách trắng (whitelist) để kiểm duyệt

```
private void createXMLStream(OutputStream outStream,
                           String quantity) throws IOException {
    // Write XML string if quantity contains numbers only.
    // Blacklisting of invalid characters can be performed
    // in conjunction.

    if (!Pattern.matches("[0-9]+", quantity)) {
        // Format violation
    }

    String xmlString = "<item>\n<description>Widget</description>\n" +
                      "<price>500</price>\n" +
                      "<quantity>" + quantity + "</quantity></item>";
    outStream.write(xmlString.getBytes());
    outStream.flush();
}
```

- Cách khác: sử dụng XML Schema

17

17

## Quy tắc IDS01-J

- Chuẩn hóa xâu trước khi kiểm duyệt để loại bỏ các mánh khoe vòng tránh (bypass) dựa trên mã ký tự
- Ví dụ: Mã nguồn có lỗi

```
// String s may be user controllable
// \uFE64 is normalized to < and \uFE65 is normalized to > using NFKC
String s = "\uFE64" + "script" + "\uFE65";

// Validate
Pattern pattern = Pattern.compile("[<>]"); // Check for angle brackets
Matcher matcher = pattern.matcher(s);
if (matcher.find()) {
    // Found black listed tag
    throw new IllegalStateException();
} else {
    // ...
}
```

➤Sửa lỗi: Thêm lệnh sau trước khi kiểm duyệt

```
s = Normalizer.normalize(s, Form.NFKC);
```

18

18

## Quy tắc IDS02-J

- Chuẩn hóa đường dẫn file trước khi kiểm duyệt để tránh lỗi hỏng duyệt thư mục hoặc truy cập vượt phạm vi.
- Ví dụ lỗi: kiểm soát không cho người dùng truy cập ngoài thư mục của họ. Tuy nhiên, mã nguồn không kiểm soát họ có thể ra ngoài bằng shortcut/symbolic link

```
public static void main(String[] args) {  
    File f = new File(System.getProperty("user.home") +  
        System.getProperty("file.separator") + args[0]);  
  
    String absPath = f.getAbsolutePath();  
  
    if (!isInSecureDir(Paths.get(absPath))) {  
        throw new IllegalArgumentException();  
    }  
    if (!validate(absPath)) { // Validation  
        throw new IllegalArgumentException();  
    }  
}
```

19

19

## Quy tắc IDS02-J: Ví dụ - Sửa lỗi

```
public static void main(String[] args) throws IOException {  
    File f = new File(System.getProperty("user.home") +  
        System.getProperty("file.separator") + args[0]);  
  
    String canonicalPath = f.getCanonicalPath();  
  
    if (!isInSecureDir(Paths.get(canonicalPath))) {  
        throw new IllegalArgumentException();  
    }  
    if (!validate(canonicalPath)) { // Validation  
        throw new IllegalArgumentException();  
    }  
}
```

20

20

## Quy tắc IDS03-J

- Làm sạch dữ liệu trước khi ghi vào file nhật ký để tránh gây hiểu nhầm

- Ví dụ lỗi:

```
if (loginSuccessful) {  
    logger.severe("User login succeeded for: " + username);  
} else {  
    logger.severe("User login failed for: " + username);  
}
```

Đối phương có thể nhập tên tài khoản như sau:

```
david  
May 15, 2011 2:25:52 PM java.util.logging.LogManager$RootLogger log  
SEVERE: User login succeeded for: administrator
```

Sửa lỗi:

```
if (!Pattern.matches("[A-Za-z0-9_]+", username)) {  
    // Unsanitized username  
    logger.severe("User login failed for unauthorized user");  
}
```

21

21

## Quy tắc IDS07-J

- Không truyền dữ liệu không đáng tin, chưa được làm sạch cho phương thức Runtime.exec() để tránh tấn công chèn lệnh thực thi
- Ví dụ lỗi: sử dụng exec() để liệt kê thư mục

```
class DirList {  
    public static void main(String[] args) throws Exception {  
        String dir = System.getProperty("dir");  
        Runtime rt = Runtime.getRuntime();  
        Process proc = rt.exec("cmd.exe /C dir " + dir);  
        int result = proc.waitFor();  
        if (result != 0) {  
            System.out.println("process error: " + result);  
        }  
        InputStream in = (result == 0) ? proc.getInputStream() :  
            proc.getErrorStream();  
        int c;  
        while ((c = in.read()) != -1) {  
            System.out.print((char) c);  
        }  
    }  
}
```

dir = "dummy & del /f \*.\*" → exec(cmd.exe /C dir dummy & del /f \*.\*)

22

22

## Quy tắc IDS07-J: Ví dụ - Sửa lỗi

- Tránh dùng Runtime.exec()

```
import java.io.File;

class DirList {
    public static void main(String[] args) throws Exception {
        File dir = new File(System.getProperty("dir"));
        if (!dir.isDirectory()) {
            System.out.println("Not a directory");
        } else {
            for (String file : dir.list()) {
                System.out.println(file);
            }
        }
    }
}
```

- Hoặc kiểm duyệt

```
// ...
if (!Pattern.matches("[0-9A-Za-z@.]+", dir)) {
    // Handle error
}
// ...
```

23

23

## Quy tắc IDS08-J

- Làm sạch dữ liệu trước khi truyền vào biểu thức chính quy(regex: regular expression) để tránh các mảnh khόé vòng tránh.

- Ví dụ lỗi:

```
public static Set<String> suggestSearches(String search) {
    synchronized(lock) {
        Set<String> searches = new HashSet<String>();

        // Construct regex dynamically from user string
        String regex = "(.*? +public\\[\\d+\\] +.*" + search + ".*)";

        Pattern keywordPattern = Pattern.compile(regex);
        Matcher logMatcher = keywordPattern.matcher(log);
        while (logMatcher.find()) {
            String found = logMatcher.group(1);
            searches.add(found);
        }
        return searches;
    }
}
```

search = “.\*”|(.\*” → regex = (.? +public[\d+] +.\*.)|(.\*.)

24

24

## Quy tắc IDS08-J: Ví dụ - Sửa lỗi

```
public static Set<String> suggestSearches(String search) {  
    synchronized(lock) {  
        Set<String> searches = new HashSet<String>();  
  
        StringBuilder sb = new StringBuilder(search.length());  
        for (int i = 0; i < search.length(); ++i) {  
            char ch = search.charAt(i);  
            if (Character.isLetterOrDigit(ch) ||  
                ch == ' ' ||  
                ch == '\'') {  
                sb.append(ch);  
            }  
        }  
        search = sb.toString();  
  
        // Construct regex dynamically from user string  
        String regex = "(.*? +public\\[\\d+] +" + search + ".*)";  
        // ...  
    }  
}
```

25

25

## Quy tắc IDS13-J

- Sử dụng bảng mã ký tự tương thích cho 2 chiều xử lý(doc-ghi, mã hóa-giải mã, gửi-nhận...) khi xử lý ký tự
- Ví dụ lỗi:

```
FileInputStream fis = null;  
try {  
    fis = new FileInputStream("SomeFile");  
    DataInputStream dis = new DataInputStream(fis);  
  
    byte[] data = new byte[1024];  
    dis.readFully(data);  
    String result = new String(data);  
} catch (IOException x) {  
    // handle error  
} finally {  
    if (fis != null) {  
        try {  
            fis.close();  
        } catch (IOException x) {  
            // Forward to handler  
        }  
    }  
}
```

- Sửa lỗi: `String result = new String(data, SOME_ENCODING);`

26

26

### 3. BIỂU THỨC

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

27

27

### Quy tắc EXP00-J

- Không bỏ qua giá trị trả về của hàm, khiến chương trình không kiểm soát được trạng thái xử lý

- Ví dụ lỗi:

```
public void deleteFile() {  
    File someFile = new File("someFileName.txt");  
    // do something with someFile  
    someFile.delete();  
}
```

- Sửa lỗi

```
public void deleteFile() {  
    File someFile = new File("someFileName.txt");  
    // do something with someFile  
    if (!someFile.delete()) {  
        // handle failure to delete the file  
    }  
}
```

- Ví dụ lỗi:

```
String original = "insecure";  
original.replace('i', '9');
```

- Sửa lỗi

```
String original = "insecure";  
original = original.replace('i', '9');
```

28

28

## Quy tắc EXP01-J

- Không bao giờ sử dụng một con trỏ null để tránh ngoại lệ gây kết thúc bất thường hoặc tấn công DoS

- Ví dụ lỗi:

```
public static int cardinality(Object obj, final Collection col) {  
    int count = 0;  
    Iterator it = col.iterator();  
    while (it.hasNext()) {  
        Object elt = it.next();  
        // null pointer dereference  
        if ((null == obj && null == elt) || obj.equals(elt)) {  
            count++;  
        }  
    }  
    return count;  
}
```

Tomcat 4.1.24

- Sửa lỗi:

```
if ((null == obj && null == elt) ||  
    (null != obj && obj.equals(elt))) {  
    count++;  
}
```

29

29

## Quy tắc EXP02-J

- Sử dụng phương thức Arrays.equal() để so sánh mảng

- Ví dụ lỗi:

```
public void arrayEqualsExample() {  
    int[] arr1 = new int[20]; // initialized to 0  
    int[] arr2 = new int[20]; // initialized to 0  
    arr1.equals(arr2); // false  
}
```

- Sửa lỗi: Arrays.equal(ar21, arr2)

- Quy tắc liên quan: EXP03-J

➢ Toán tử == để so sánh 2 giá trị có kiểu nguyên thủy

➢ Phương thức equal() để so sánh 2 đối tượng

30

30

## Quy tắc EXP04-J

- Gán giá trị nguyên thủy có kiểu phù hợp với lớp bao.
- Ví dụ lỗi:

```
public class ShortSet {  
    public static void main(String[] args) {  
        HashSet<Short> s = new HashSet<Short>();  
        for (short i = 0; i < 100; i++) {  
            s.add(i);  
            s.remove(i - 1); // tries to remove an Integer  
        }  
        System.out.println(s.size());  
    }  
}
```

- Sửa lỗi: `s.remove((short) i-1)`

31

31

## 4. KIỂU DỮ LIỆU SỐ VÀ CÁC TOÁN TỬ

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

32

32

## Quy tắc NUM00-J

- Phát hiện hoặc phòng tránh lỗi tràn số nguyên
- Kiểu dữ liệu số nguyên trong Java có những giới hạn biểu diễn khác nhau
- Lỗi tràn số nguyên(Integer Overflow): giá trị vượt ra khỏi dải biểu diễn của kiểu dữ liệu số nguyên
- Cách thức phòng tránh:
  - Kiểm thử tiền điều kiện
  - Nâng kiểu (upcasting)
  - Sử dụng lớp BigInteger

33

33

## Sử dụng tiền điều kiện

- Kiểm tra giá trị toán hạng trước khi thực hiện toán tử
- Hạn chế: Khó triển khai do các toán tử khác nhau thì điều kiện cho toán hạng cũng khác nhau.
- Ví dụ:

```
static final int safeAdd(int left, int right)
    throws ArithmeticException {
    if (right > 0 ? left > Integer.MAX_VALUE - right
        : left < Integer.MIN_VALUE - right) {
        throw new ArithmeticException("Integer overflow");
    }
    return left + right;
}
```

- Hãy viết phương thức bao cho các toán tử khác

34

34

## Sử dụng nâng kiểu

- Có thể thực hiện nâng kiểu dữ liệu theo thứ tự sau để mở rộng dải biểu diễn:

byte → short → int → long

- Hạn chế: không thể nâng kiểu cho giá trị kiểu long

- Ví dụ:

```
public static long intRangeCheck(long value)
    throws ArithmeticException {
    if ((value < Integer.MIN_VALUE) || (value > Integer.MAX_VALUE)) {
        throw new ArithmeticException("Integer overflow");
    }
    return value;
}

public static int multAccum(int oldAcc, int newVal, int scale)
    throws ArithmeticException {
    final long res = intRangeCheck(
        ((long) oldAcc) + intRangeCheck((long) newVal * (long) scale)
    );
    return (int) res; // safe down-cast
}
```

35

35

## Sử dụng BigInteger

- Tương tự kỹ thuật nâng kiểu dữ liệu

- Ví dụ:

```
private static final BigInteger bigMaxInt =
    BigInteger.valueOf(Integer.MAX_VALUE);
private static final BigInteger bigMinInt =
    BigInteger.valueOf(Integer.MIN_VALUE);

public static BigInteger intRangeCheck(BigInteger val)
    throws ArithmeticException {
    if (val.compareTo(bigMaxInt) == 1 ||
        val.compareTo(bigMinInt) == -1) {
        throw new ArithmeticException("Integer overflow");
    }
    return val;
}

public static int multAccum(int oldAcc, int newVal, int scale)
    throws ArithmeticException {
    BigInteger product =
        BigInteger.valueOf(newVal).multiply(BigInteger.valueOf(scale));
    BigInteger res =
        intRangeCheck(BigInteger.valueOf(oldAcc).add(product));
    return res.intValue(); // safe conversion
}
```

36

36

## Quy tắc NUM02-J

- Đảm bảo các phép chia không xảy ra lỗi chia cho 0

37

37

## Quy tắc NUM03-J

- Sử dụng kiểu số nguyên phù hợp để có thể biểu diễn giá trị số nguyên không dấu lớn nhất
- Ví dụ lỗi:

```
public static int getInteger(DataInputStream is) throws IOException {
    return is.readInt();
}
```

- Sửa lỗi:

```
public static long getInteger(DataInputStream is) throws IOException {
    return is.readInt() & 0xFFFFFFFFL; // mask with 32 one-bits
}
```

38

38

## Quy tắc NUM08-J

- Kiểm tra số thực dấu phẩy động cho các giá trị đặc biệt (infinity, -infinity, NaN)
- Ví dụ lỗi:

```
double currentBalance; // User's cash balance
void doDeposit(String userInput) {
    double val;
    try {
        val = Double.valueOf(userInput);
    } catch (NumberFormatException e) {
        // Handle input format error
    }
    if (val >= Double.MAX_VALUE - currentBalance) {
        // Handle range error
    }
    currentBalance += val;
}
```

39

39

## Quy tắc NUM08-J: Ví dụ - Sửa lỗi

```
double currentBalance; // User's cash balance
void doDeposit(String s){
    double val;
    try {
        val = Double.valueOf(userInput);
    } catch (NumberFormatException e) {
        // Handle input format error
    }
    if (Double.isInfinite(val)){
        // Handle infinity error
    }
    if (Double.isNaN(val)) {
        // Handle NaN error
    }
    if (val >= Double.MAX_VALUE - currentBalance) {
        // Handle range error
    }
    currentBalance += val;
}
```

40

40

## Quy tắc NUM09-J

- Không sử dụng biến số thực cho biến đếm trong vòng lặp

- Ví dụ lỗi:

```
for (float x = 0.1f; x <= 1.0f; x += 0.1f) {  
    System.out.println(x);  
}
```

- Sửa lỗi

```
for (int count = 1; count <= 10; count += 1) {  
    float x = count/10.0f;  
    System.out.println(x);  
}
```

- Ví dụ lỗi:

```
for (float x = 100000001.0f; x <= 100000010.0f; x += 1.0f)
```

- Sửa lỗi:

```
for (int count = 1; count <= 10; count += 1) {  
    double x = 100000000.0 + count;  
    /* ... */  
}
```

41

41

## Quy tắc NUM12-J

- Đảm bảo không mất mát khi chuyển đổi kiểu dữ liệu cho giá trị xuống kiểu thấp hơn(có dải biểu diễn hẹp hơn)

- Dải biểu diễn tăng rộng theo thứ tự sau:

- byte
- short, char
- int
- long
- float
- double

- Quy tắc NUM13-J: tránh làm mất độ chính xác khi chuyển từ kiểu dữ liệu số nguyên thành số thực

42

42

## 5. CÁC QUY TẮC HƯỚNG ĐỐI TƯỢNG

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

43

43

### Quy tắc OBJOO-J

- Chỉ mở rộng lớp và các phương thức tới các lớp con tin cậy
- Java cho phép các lớp con ghi đè lên các thuộc tính, phương thức kế thừa từ lớp cha
- Sử dụng từ khóa `final` khi khai báo các thành viên bắt biến để ngăn cản ghi đè nhưng hạn chế khả năng mở rộng của lớp.
- Vấn đề bảo mật: các lớp vừa phải cho phép các lớp con mở rộng, vừa phải chống lại việc mở rộng bởi mã nguồn không đáng tin

44

44

## Quy tắc OBJ00-J: Ví dụ 1

```
BigInteger msg = new BigInteger("123");
msg = msg.modPow(exp, m); // Always returns 1

// Malicious subclassing of java.math.BigInteger
class BigInteger extends java.math.BigInteger {
    private int value;

    public BigInteger(String str) {
        super(str);
        value = Integer.parseInt(str);
    }

    public void setValue(int value) {
        this.value = value;
    }

    @Override public java.math.BigInteger modPow(
        java.math.BigInteger exponent, java.math.BigInteger m) {
        this.value = ((int) (Math.pow(this.doubleValue(),
            exponent.doubleValue()))) % m.intValue();
        return this;
    }
}
```

45

45

## Quy tắc OBJ00-J: Ví dụ 1 – Sửa lỗi

- Kiểm soát phương thức khởi tạo không phải là một giải pháp

```
public class BigInteger {
    public BigInteger(String str) {
        // java.lang.Object.getClass(), which is final
        Class c = getClass();
        // Confirm class type
        if (c != java.math.BigInteger.class) {
            // Check the permission needed to subclass BigInteger
            // throws a security exception if not allowed
            securityManagerCheck();
        }
        // ...
    }
}
```

46

46

## Quy tắc OBJ00-J: Ví dụ 1 - Sửa lỗi

- Làm sạch lớp khi khởi tạo

```
public static BigInteger safeInstance(BigInteger val) {  
    // create a defensive copy if it is not java.math.BigInteger  
    if (val.getClass() != java.math.BigInteger.class) {  
        return new BigInteger(val.toByteArray());  
    }  
    return val;  
}
```

47

47

## Quy tắc OBJ00-J: Ví dụ 1 - Sửa lỗi

- Sử dụng phương thức khởi tạo private để bảo vệ lớp cha

```
public class BigInteger {  
    public BigInteger(String str) {  
        // throws a security exception if not allowed  
        this(str, check(this.getClass()));  
    }  
  
    private BigInteger(String str, boolean securityManagerCheck) {  
        // regular construction goes here  
    }  
  
    private static boolean check(Class c) {  
        // Confirm class type  
        if (c != java.math.BigInteger.class) {  
            // Check the permission needed to subclass BigInteger  
            // throws a security exception if not allowed  
            securityManagerCheck();  
        }  
        return true;  
    }  
}
```

48

48

## Quy tắc OBJ00-J: Ví dụ 2

- Phương thức `getMethodName()` bị ghi đè cho phép gọi phương thức bất kỳ

```
public class MethodInvoker {  
    public void invokeMethod() {  
        AccessController.doPrivileged(new PrivilegedAction<Object>() {  
            public Object run() {  
                try {  
                    Class<?> thisClass = MethodInvoker.class;  
                    String methodName = getMethodName();  
                    Method method = thisClass.getMethod(methodName, null);  
                    method.invoke(new MethodInvoker(), null);  
                } catch (Throwable t) {  
                    // Forward to handler  
                }  
                return null;  
            }  
        );  
    }  
  
    String getMethodName() {  
        return "someMethod";  
    }  
  
    public void someMethod() {  
        // ...  
    }  
    // Other methods  
}
```

Sửa lỗi: `String methodName = MethodInvoker.this.getMethodName();`

49

49

## Quy tắc OBJ01-J

- Khai báo thuộc tính với chỉ định truy cập là `private` và cung cấp các phương thức để truy cập(`getter`, `setter`)
- Lợi ích:
  - Có thể kiểm duyệt dữ liệu đầu vào khi ghi bằng `setter` và lọc giá trị đầu ra khi đọc bằng `getter`
  - Che giấu biểu diễn trạng thái bên trong, sử dụng biểu diễn thay thế khi cung cấp dữ liệu ra bên ngoài
  - Có thể thiết lập chỉ định truy cập cho `getter` và `setter` là khác nhau
  - Các lợi ích khác cho quá trình phát triển phần mềm

50

50

## Quy tắc OBJ02-J

- Bảo toàn ràng buộc triển khai trên lớp con khi cập nhật lớp cha

```
private class Account {  
    // Maintains all banking related data such as account balance  
    private double balance = 100;  
  
    boolean withdraw(double amount) {  
        if ((balance - amount) >= 0) {  
            balance -= amount;  
            System.out.println("Withdrawal successful. The balance is : "  
                + balance);  
            return true;  
        }  
        return false;  
    }  
  
    public class BankAccount extends Account {  
        // Subclass handles authentication  
        @Override boolean withdraw(double amount) {  
            if (!securityCheck()) {  
                throw new IllegalAccessException();  
            }  
            return super.withdraw(amount);  
        }  
        private final boolean securityCheck() {  
            // check that account management may proceed  
        }  
    }  
}
```

51

51

## Quy tắc OBJ02-J: Ví dụ lỗi

```
private class Account {  
    // Maintains all banking related data such as account balance  
    boolean overdraft() {  
        balance += 300; // Add 300 in case there is an overdraft  
        System.out.println("Added back-up amount. The balance is : "  
            + balance);  
        return true;  
    }  
  
    // other Account methods  
}  
  
public class BankAccount extends Account {  
    // Subclass handles authentication  
    // NOTE: unchanged from previous version  
    // NOTE: lacks override of overdraft method  
}
```

52

52

## Quy tắc OBJ03-J

- Không trộn lẫn kiểu dữ liệu tổng quát và không tổng quát
- Ví dụ lỗi: Khi thực thi sẽ phát sinh ngoại lệ

```
class ListUtility {  
    private static void addToList(List list, Object obj) {  
        list.add(obj); // Unchecked warning  
    }  
  
    public static void main(String[] args) {  
        List<String> list = new ArrayList<String>();  
        addToList(list, 1);  
        System.out.println(list.get(0));  
    }  
}
```

- Sửa lỗi:

```
class ListUtility {  
    private static void addToList(List<String> list, String str) {  
        list.add(str); // No warning generated  
    }  
  
    public static void main(String[] args) {  
        List<String> list = new ArrayList<String>();  
        addToList(list, "1");  
        System.out.println(list.get(0));  
    }  
}
```

3

53

## Quy tắc OBJ03-J: Lỗi khác

```
class ListAdder {  
    @SuppressWarnings("unchecked")  
    private static void addToList(List list, Object obj) {  
        list.add(obj); // Unchecked warning  
    }  
  
    private static <T> void printOne(T type) {  
        if (!(type instanceof Integer || type instanceof Double)) {  
            System.out.println("Cannot print in the supplied type");  
        }  
        List<T> list = new ArrayList<T>();  
        addToList(list, 1);  
        System.out.println(list.get(0));  
    }  
  
    public static void main(String[] args) {  
        double d = 1;  
        int i = 1;  
        System.out.println(d);  
        ListAdder.printOne(d);  
        System.out.println(i);  
        ListAdder.printOne(i);  
    }  
}
```

54

54

## Quy tắc OBJ03-J: Lỗi khác – Sửa lỗi

```
class ListAdder {  
    private static <T> void addToList(List<T> list, T t) {  
        list.add(t); // No warning generated  
    }  
  
    private static <T> void printOne(T type) {  
        if (type instanceof Integer) {  
            List<Integer> list = new ArrayList<Integer>();  
            addToList(list, 1);  
            System.out.println(list.get(0));  
        } else if (type instanceof Double) {  
            List<Double> list = new ArrayList<Double>();  
  
            // This will not compile if addToList(list, 1) is used  
            addToList(list, 1.0);  
            System.out.println(list.get(0));  
        } else {  
            System.out.println("Cannot print in the supplied type");  
        }  
    }  
  
    public static void main(String[] args) {  
        double d = 1;  
        int i = 1;  
        System.out.println(d);  
        ListAdder.printOne(d);  
        System.out.println(i);  
        ListAdder.printOne(i);  
    }  
}
```

55

55

## Quy tắc OBJ04-J

- Các lớp khả biến, mà có cung cấp phương thức thay đổi thuộc tính của nó, cần có chức năng sao chép để an toàn khi truyền thể hiện sang thành phần không đáng tin cậy.
- Ví dụ lỗi:

```
public final class MutableClass {  
    private Date date;  
  
    public MutableClass(Date d) {  
        this.date = d;  
    }  
    public void setDate(Date d) {  
        this.date = d;  
    }  
  
    public Date getDate() {  
        return date;  
    }  
}
```

56

56

## Quy tắc OBJ04-J: Ví dụ - Sửa lỗi

- Cách thức 1

```
public final class MutableClass { // Copy Constructor
    private final Date date;

    public MutableClass(MutableClass mc) {
        this.date = new Date(mc.date.getTime());
    }

    public MutableClass(Date d) {
        this.date = new Date(d.getTime()); // Make defensive copy
    }

    public Date getDate() {
        return (Date) date.clone(); // Copy and return
    }
}
```

57

57

## Quy tắc OBJ04-J: Ví dụ - Sửa lỗi

- Cách thức 2

```
class MutableClass {
    private final Date date;
    private MutableClass(Date d) {
        // Noninstantiable and nonsubclassable
        this.date = new Date(d.getTime()); // Make defensive copy
    }

    public Date getDate() {
        return (Date) date.clone(); // Copy and return
    }

    public static MutableClass getInstance(MutableClass mc) {
        return new MutableClass(mc.getDate());
    }
}
```

58

58

## Quy tắc OBJ04-J: Ví dụ - Sửa lỗi

- Cách thức 3

```
public final class MutableClass implements Cloneable {  
    private Date date;  
  
    public MutableClass(Date d) {  
        this.date = new Date(d.getTime());  
    }  
  
    public Date getDate() {  
        return (Date) date.clone();  
    }  
  
    public void setDate(Date d) {  
        this.date = (Date) d.clone();  
    }  
  
    public Object clone() throws CloneNotSupportedException {  
        final MutableClass cloned = (MutableClass) super.clone();  
        // manually copy mutable Date object  
        cloned.date = (Date) date.clone();  
        return cloned;  
    }  
}
```

59

59

## Quy tắc OBJ04-J: Ví dụ - Sửa lỗi

- Cách thức 4

```
public final class MutableClass implements Cloneable {  
    private final Date date; // final field  
  
    public MutableClass(Date d) {  
        this.date = new Date(d.getTime()); // copy-in  
    }  
  
    public Date getDate() {  
        return (Date) date.clone(); // copy and return  
    }  
  
    public Object clone() {  
        Date d = (Date) date.clone();  
        MutableClass cloned = new MutableClass(d);  
        return cloned;  
    }  
}
```

60

60

## Quy tắc OBJ04-J: Ví dụ - Sửa lỗi

- Cách thức 5

```
class UnmodifiableDateView extends Date {  
    private Date date;  
  
    public UnmodifiableDateView(Date date) {  
        this.date = date;  
    }  
  
    public void setTime(long date) {  
        throw new UnsupportedOperationException();  
    }  
  
    // Override all other mutator methods  
    // to throw UnsupportedOperationException  
}  
  
public final class MutableClass {  
    private Date date;  
  
    public MutableClass(Date d) {  
        this.date = d;  
    }  
  
    public void setDate(Date d) {  
        this.date = (Date) d.clone();  
    }  
  
    public UnmodifiableDateView getDate() {  
        return new UnmodifiableDateView(date);  
    }  
}
```

61

61

## Quy tắc OBJ05-J

- Các lớp khả biến chỉ nên cung cấp bản sao tham chiếu của thuộc tính private.
- Ví dụ lỗi:

```
class MutableClass {  
    private Date d;  
  
    public MutableClass() {  
        d = new Date();  
    }  
  
    public Date getDate() {  
        return d;  
    }  
}
```

- Sửa lỗi:

```
public Date getDate() {  
    return (Date)d.clone();  
}
```

62

62

## Quy tắc OBJ05-J: Ví dụ 2

```
class MutableClass {  
    private Date[] date;  
  
    public MutableClass() {  
        date = new Date[20];  
        for (int i = 0; i < date.length; i++) {  
            date[i] = new Date();  
        }  
    }  
  
    public Date[] getDate() {  
        return date; // or return date.clone()  
    }  
}
```

63

63

## Quy tắc OBJ05-J: Ví dụ 2 – Sửa lỗi

- Kỹ thuật Deep Copy

```
class MutableClass {  
    private Date[] date;  
  
    public MutableClass() {  
        date = new Date[20];  
        for(int i = 0; i < date.length; i++) {  
            date[i] = new Date();  
        }  
    }  
  
    public Date[] getDate() {  
        Date[] dates = new Date[date.length];  
        for (int i = 0; i < date.length; i++) {  
            dates[i] = (Date) date[i].clone();  
        }  
        return dates;  
    }  
}
```

64

64

## Quy tắc OBJ05-J: Ví dụ 3

```
class ReturnRef {  
    // Internal state, may contain sensitive data  
    private Hashtable<Integer, String> ht =  
        new Hashtable<Integer, String>();  
  
    private ReturnRef() {  
        ht.put(1, "123-45-6666");  
    }  
  
    public Hashtable<Integer, String> getValues() {  
        return ht;  
    }  
  
    public static void main(String[] args) {  
        ReturnRef rr = new ReturnRef();  
        // Prints sensitive data 123-45-6666  
        Hashtable<Integer, String> ht1 = rr.getValues();  
        // Untrusted caller can remove entries  
        ht1.remove(1);  
        // Now prints null, original entry is removed  
        Hashtable<Integer, String> ht2 = rr.getValues();  
    }  
}
```

- Sửa lỗi bằng kỹ thuật Shallow Copy

```
private Hashtable<Integer, String> getValues() {  
    return (Hashtable<Integer, String>) ht.clone(); // shallow copy  
}
```

65

65

## Quy tắc OBJ06-J

- Sử dụng bản sao cho dữ liệu đầu vào và các thành phần bên trong nó có thể bị thay đổi
- Tránh lỗ hổng điều kiện tranh đua (Race Condition)
- Ví dụ lỗi

```
public final class MutableDemo {  
    // java.net.HttpCookie is mutable  
    public void useMutableInput(HttpCookie cookie) {  
        if (cookie == null) {  
            throw new NullPointerException();  
        }  
  
        // Check whether cookie has expired  
        if (cookie.hasExpired()) {  
            // Cookie is no longer valid,  
            // handle condition by throwing an exception  
        }  
  
        // Cookie may have expired since time of check  
        doLogic(cookie);  
    }  
}
```

66

66

## Quy tắc OBJ06-J: Ví dụ - Sửa lỗi

```
public final class MutableDemo {  
    // java.net.HttpCookie is mutable  
    public void useMutableInput(HttpCookie cookie) {  
        if (cookie == null) {  
            throw new NullPointerException();  
        }  
  
        // Create copy  
        cookie = (HttpCookie)cookie.clone();  
  
        // Check whether cookie has expired  
        if (cookie.hasExpired()) {  
            // Cookie is no longer valid.  
            // handle condition by throwing an exception  
        }  
  
        doLogic(cookie);  
    }  
}
```

67

67

## Quy tắc OBJ06-J: Ví dụ - Sửa lỗi

- Kết hợp Shallow Copy và Deep Copy với mảng không phải kiểu nguyên thủy

```
public void deepCopy(int[] ints, HttpCookie[] cookies) {  
    if (ints == null || cookies == null) {  
        throw new NullPointerException();  
    }  
  
    // Shallow copy  
    int[] intsCopy = ints.clone();  
  
    // Deep copy  
    HttpCookie[] cookiesCopy = new HttpCookie[cookies.length];  
    for (int i = 0; i < cookies.length; i++) {  
        // Manually create copy of each element in array  
        cookiesCopy[i] = (HttpCookie)cookies[i].clone();  
    }  
  
    doLogic(intsCopy, cookiesCopy);  
}
```

68

68

## Quy tắc OBJ07-J

- Không cho phép sao chép đối tượng của các lớp chứa dữ liệu nhạy cảm

➤ Việc sao chép bỏ qua các kiểm soát ATBM ở phương thức khởi tạo

```
class SensitiveClass {  
    private char[] filename;  
    private Boolean shared = false;  
  
    SensitiveClass(String filename) {  
        this.filename = filename.toCharArray();  
    }  
  
    final void replace() {  
        if (!shared) {  
            for(int i = 0; i < filename.length; i++) {  
                filename[i] = 'x';  
            }  
        }  
    }  
    final String get() {  
        if (!shared) {  
            shared = true;  
            return String.valueOf(filename);  
        } else {  
            throw new IllegalStateException("Failed to get instance");  
        }  
    }  
  
    final void printFilename() {  
        System.out.println(String.valueOf(filename));  
    }  
}
```

69

69

## Quy tắc OBJ07-J: Ví dụ lỗi – Tấn công

```
class MaliciousSubclass extends SensitiveClass implements Cloneable {  
    protected MaliciousSubclass(String filename) {  
        super(filename);  
    }  
  
    @Override public MaliciousSubclass clone() {  
        // Well-behaved clone() method  
        MaliciousSubclass s = null;  
        try {  
            s = (MaliciousSubclass)super.clone();  
        } catch(Exception e) {  
            System.out.println("not cloneable");  
        }  
        return s;  
    }  
  
    public static void main(String[] args) {  
        MaliciousSubclass ms1 = new MaliciousSubclass("file.txt");  
        MaliciousSubclass ms2 = ms1.clone(); // Creates a copy  
        String s = ms1.get(); // Returns filename  
        System.out.println(s); // Filename is "file.txt"  
        ms2.replace(); // Replaces all characters with 'x'  
        // Both ms1.get() and ms2.get() will subsequently  
        // return filename = 'xxxxxxxx'  
        ms1.printFilename(); // Filename becomes 'xxxxxxxx'  
        ms2.printFilename(); // Filename becomes 'xxxxxxxx'  
    }  
}
```

70

70

## Quy tắc OBJ07-J: Ví dụ lỗi – Sửa lỗi

- Cách thức 1: Không cho kế thừa

```
final class SensitiveClass {  
    // ...  
}
```

- Cách thức 2: Phương thức clone() phải là bắt buộc

```
class SensitiveClass {  
    // ...  
    public final SensitiveClass clone()  
        throws CloneNotSupportedException {  
        throw new CloneNotSupportedException();  
    }  
}
```

71

71

## Quy tắc OBJ08-J

- Không bọc lộ thành viên private ra lớp ngoài thông qua một lớp lồng(nested class)
- Lớp lồng là lớp được khai báo bên trong một lớp khác(lớp ngoài)

➤ Lớp lồng có thể truy cập tới bất kỳ thành viên nào của lớp ngoài

- Ví dụ lỗi:

```
class Coordinates {  
    private int x;  
    private int y;  
  
    public class Point {  
        public void getPoint() {  
            System.out.println("(" + x + "," + y + ")");  
        }  
    }  
}
```

- Sửa lỗi: sử dụng chỉ định truy cập private cho lớp lồng

72

72

## Quy tắc OBJ10-J

- Các thuộc tính có chỉ định truy cập `public` là `static` thì cũng nên là `final`
- Ví dụ lỗi:

```
package org.apache.xpath.compiler;  
  
public class FunctionTable {  
    public static FuncLoader m_functions;  
}
```

*Lỗi trong Java JDK 1.4.2*

- Sửa lỗi:

```
public static final FuncLoader m_functions;  
// Initialize m_functions in a constructor
```

73

73

## 6. CÁC QUY TẮC VỚI PHƯƠNG THỨC

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

74

74

## Quy tắc METOO-J

- Kiểm duyệt tham số truyền cho phương thức để tránh truy cập bộ nhớ không hợp lệ
- Ví dụ lỗi:

```
private Object myState = null;

// Sets some internal state in the library
void setState(Object state) {
    myState = state;
}

// Performs some action using the file passed earlier
void useState() {
    // Perform some action here
}
```

75

75

## Quy tắc METOO-J: Ví dụ - Sửa lỗi

```
private Object myState = null;

// Sets some internal state in the library
void setState(Object state) {
    if (state == null) {
        // Handle null state
    }

    // Defensive copy here when state is mutable
    if (isValidState(state)) {
        // Handle invalid state
    }
    myState = state;
}

// Performs some action using the state passed earlier
void useState() {
    if (myState == null) {
        // Handle no state (e.g. null) condition
    }
    //...
}
```

76

76

## Quy tắc MET02-J

- Không sử dụng các phương thức hoặc lớp lỗi thời(deprecated)

77

77

## Quy tắc MET03-J

- Phương thức thực hiện kiểm tra bảo mật thì phải là **private** hoặc **final** để tránh bị lớp con ghi đè

- Ví dụ lỗi:

```
public void readSensitiveFile() {
    try {
        SecurityManager sm = System.getSecurityManager();
        if (sm != null) { // Check for permission to read file
            sm.checkRead("/temp/tempFile");
        }
        // Access the file
    } catch (SecurityException se) {
        // Log exception
    }
}
```

- Sửa lỗi:

```
public final void readSensitiveFile()
```

- Hoặc:

```
private void readSensitiveFile()
```

78

78

## Quy tắc MET04-J

- Không nói chỉ định truy cập khi thực hiện ghi đè phương thức.

- Ví dụ lỗi:

```
class Super {  
    protected void doLogic() {  
        System.out.println("Super invoked");  
    }  
  
    public class Sub extends Super {  
        public void doLogic() {  
            System.out.println("Sub invoked");  
            // Do sensitive operations  
        }  
    }  
}
```

- Sửa lỗi:

```
class Super {  
    protected final void doLogic() { // declare as final  
        System.out.println("Super invoked");  
        // Do sensitive operations  
    }  
}
```

79

79

## Quy tắc MET05-J

- Đảm bảo các phương thức khởi tạo không gọi các phương thức có thể bị ghi đè

- Ví dụ lỗi:

```
class SuperClass {  
    public SuperClass () {  
        doLogic();  
    }  
  
    public void doLogic() {  
        System.out.println("This is superclass!");  
    }  
  
    class SubClass extends SuperClass {  
        private String color = null;  
        public SubClass() {  
            super();  
            color = "Red";  
        }  
  
        public void doLogic() {  
            System.out.println("This is subclass! The color is :" + color);  
            // ...  
        }  
    }  
  
    public class Overridable {  
        public static void main(String[] args) {  
            SuperClass bc = new SuperClass();  
            // Prints "This is superclass!"  
            SuperClass sc = new SubClass();  
            // Prints "This is subclass! The color is :null"  
        }  
    }  
}
```

80

80

## Quy tắc MET05-J: Ví dụ - Sửa lỗi

```
class SuperClass {  
    public SuperClass() {  
        doLogic();  
    }  
    public final void doLogic() {  
        System.out.println("This is superclass!");  
    }  
}
```

81

81

## Quy tắc MET06-J

- Không gọi các phương thức có thể bị ghi đè trong phương thức clone()
- Ví dụ lỗi

```
public Object clone() throws CloneNotSupportedException {  
    final CloneExample clone = (CloneExample) super.clone();  
    clone.doSomething(); // Invokes overridable method  
    clone.cookies = clone.deepCopy();  
    return clone;  
}  
  
void doSomething() { // Overridable  
    for (int i = 0; i < cookies.length; i++) {  
        cookies[i].setValue("'" + i);  
    }  
}  
HttpCookie[] deepCopy() {  
    if (cookies == null) {  
        throw new NullPointerException();  
    }  
  
    // deep copy  
    HttpCookie[] cookiesCopy = new HttpCookie[cookies.length];  
  
    for (int i = 0; i < cookies.length; i++) {  
        // Manually create a copy of each element in array  
        cookiesCopy[i] = (HttpCookie) cookies[i].clone();  
    }  
    return cookiesCopy;  
}
```

82

82

## Quy tắc MET10-J

- Khi triển khai phương thức compareTo() cần phải tuân thủ các ràng buộc thông thường
- Một số ràng buộc:
  - $\text{sgn}(x.\text{compareTo}(y)) == -\text{sgn}(y.\text{compareTo}(x))$
  - Nếu  $(x.\text{compareTo}(y) > 0 \ \&\& y.\text{compareTo}(z) > 0)$  thì  $x.\text{compareTo}(z) > 0$ .
  - Nếu  $x.\text{compareTo}(y) == 0$  thì  $\text{sgn}(x.\text{compareTo}(z)) == \text{sgn}(y.\text{compareTo}(z))$
- Không bắt buộc nhưng nên thực hiện
  - $(x.\text{compareTo}(y) == 0) == x.equals(y)$

83

83

## Quy tắc MET11-J

- Đảm bảo các giá trị khóa, mà được sử dụng trong toán tử so sánh, là không thể bị thay đổi

- Ví dụ lỗi:

```
class Employee {  
    private String name;  
    private double salary;  
  
    Employee(String empName, double empSalary) {  
        this.name = empName;  
        this.salary = empSalary;  
    }  
  
    public void setEmployeeName(String empName) {  
        this.name = empName;  
    }  
  
    public void Salary(double empSalary) {  
        this.salary = empSalary;  
    }  
  
    @Override  
    public boolean equals(Object o) {  
        if (!(o instanceof Employee)) {  
            return false;  
        }  
  
        Employee emp = (Employee)o;  
        return emp.name.equals(name);  
    }  
    public int hashCode() /* ... */  
}  
  
// Client code  
Map<Employee, Calendar> map =  
    new ConcurrentHashMap<Employee, Calendar>();  
// ...
```

84

84

## Quy tắc MET11-J: Ví dụ - Sửa lỗi

```
// Mutable class Employee
class Employee {
    private String name;
    private double salary;
    private final long employeeID; // Unique for each Employee

    Employee(String name, double salary, long empID) {
        this.name = name;
        this.salary = salary;
        this.employeeID = empID;
    }

    // ... other methods

    @Override
    public boolean equals(Object o) {
        if (!(o instanceof Employee)) {
            return false;
        }

        Employee emp = (Employee)o;
        return emp.employeeID == employeeID;
    }

    // Client code remains same
    Map<Employee, Calendar> map =
        new ConcurrentHashMap<Employee, Calendar>();
    // ...
}
```

85

85

## 7. CÁC QUY TẮC VỚI NGOẠI LỆ

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

86

86

## Quy tắc ERR00-J

- Không bỏ qua ngoại lệ

- Ví dụ lỗi:

```
try {
    //...
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

- Sửa lỗi:

```
boolean volatile validFlag = false;
do {
    try {
        // If requested file does not exist, throws FileNotFoundException
        // If requested file exists, sets validFlag to true
        validFlag = true;
    } catch (FileNotFoundException e) {
        // Ask the user for a different file name
    }
} while (validFlag != true);
// Use the file
```

87

87

## Quy tắc ERR00-J: Báo cáo ngoại lệ

```
public interface Reporter {
    public void report(Throwable t);
}

public class ExceptionReporter {

    // Exception reporter that prints the exception
    // to the console (used as default)
    private static final Reporter PrintException = new Reporter() {
        public void report(Throwable t) {
            System.err.println(t.toString());
        }
    };

    // Stores the default reporter.
    // The default reporter can be changed by the user.
    private static Reporter Default = PrintException;

    // Helps change the default reporter back to
    // PrintException in the future
    public static Reporter getPrintException() {
        return PrintException;
    }

    public static Reporter getExceptionReporter() {
        return Default;
    }

    // May throw a SecurityException (which is unchecked)
    public static void setExceptionReporter(Reporter reporter) {
        // Custom permission
        ExceptionReporterPermission perm = new
            ExceptionReporterPermission("exc.reporter");
        SecurityManager sm = System.getSecurityManager();
        if (sm != null) {
            // Check whether the caller has appropriate permissions
            sm.checkPermission(perm);
        }
        // Change the default exception reporter
        Default = reporter;
    }
}
```

88

88

## Quy tắc ERR01-J

- Không để lộ thông tin nhạy cảm trong thông báo ngoại lệ

Exception Name	Description of Information Leak or Threat
java.io.FileNotFoundException	Underlying file system structure, user name enumeration
java.sql.SQLException	Database structure, user name enumeration
java.net.BindException	Enumeration of open ports when untrusted client can choose server port
java.util.ConcurrentModificationException	May provide information about thread-unsafe code
javax.naming.InsufficientResourcesException	Insufficient server resources (may aid DoS)
java.util.MissingResourceException	Resource enumeration
java.util.jar.JarException	Underlying file system structure
java.security.acl.NotOwnerException	Owner enumeration
java.lang.OutOfMemoryError	DoS
java.lang.StackOverflowError	DoS

89

89

## Quy tắc ERR01-J: Ví dụ lỗi

```
class ExceptionExample {
    public static void main(String[] args) throws FileNotFoundException {
        // Linux stores a user's home directory path in
        // the environment variable $HOME, Windows in %APPDATA%
        FileInputStream fis =
            new FileInputStream(System.getenv("APPDATA") + args[0]);
    }
}

try {
    FileInputStream fis =
        new FileInputStream(System.getenv("APPDATA") + args[0]);
} catch (FileNotFoundException e) {
    // Log the exception
    throw new IOException("Unable to retrieve file", e);
}

class SecurityIOException extends IOException {/* ... */};

try {
    FileInputStream fis =
        new FileInputStream(System.getenv("APPDATA") + args[0]);
} catch (FileNotFoundException e) {
    // Log the exception
    throw new SecurityIOException();
}
```

90

90

## Quy tắc ERR01-J: Sửa lỗi

- Cách thức 1: Hạn chế truy cập trong thư mục chỉ định

```
class ExceptionExample {  
    public static void main(String[] args) {  
  
        File file = null;  
        try {  
            file = new File(System.getenv("APPDATA") +  
                args[0]).getCanonicalFile();  
            if (!file.getPath().startsWith("c:\\\\homepath")) {  
                System.out.println("Invalid file");  
                return;  
            }  
            } catch (IOException x) {  
                System.out.println("Invalid file");  
                return;  
            }  
  
        try {  
            FileInputStream fis = new FileInputStream(file);  
        } catch (FileNotFoundException x) {  
            System.out.println("Invalid file");  
            return;  
        }  
    }  
}
```

91

91

## Quy tắc ERR01-J: Sửa lỗi

- Cách thức 1: Kiểm soát giá trị đầu vào

```
class ExceptionExample {  
    public static void main(String[] args) {  
        FileInputStream fis = null;  
        try {  
            switch(Integer.valueOf(args[0])) {  
                case 1:  
                    fis = new FileInputStream("c:\\\\homepath\\\\file1");  
                    break;  
                case 2:  
                    fis = new FileInputStream("c:\\\\homepath\\\\file2");  
                    break;  
                //...  
                default:  
                    System.out.println("Invalid option");  
                    break;  
            }  
        } catch (Throwable t) {  
            MyExceptionReporter.report(t); // Sanitize  
        }  
    }  
}
```

92

92

## Quy tắc ERR02-J

- Phòng ngừa ngoại lệ xảy ra khi đang ghi log
- Ví dụ lỗi:

```
try {
    ...
} catch (SecurityException se) {
    System.err.println(e);
    // Recover from exception
}
```

- Sửa lỗi: Sử dụng lớp java.util.logging.Logger

```
try {
    ...
} catch(SecurityException se) {
    logger.log(Level.SEVERE, se);
    // Recover from exception
}
```

93

93

## Quy tắc ERR03-J

- Khôi phục trạng thái đối tượng trong phương thức gây ra ngoại lệ
- Ví dụ lỗi:

```
protected int getVolumePackage(int weight) {
    length += PADDING;
    width += PADDING;
    height += PADDING;
    try {
        if (length <= PADDING || width <= PADDING
            || height <= PADDING || length > MAX_DIMENSION + PADDING
            || width > MAX_DIMENSION + PADDING ||
            height > MAX_DIMENSION + PADDING || weight <= 0 ||
            weight > 20) {
            throw new IllegalArgumentException();
        }
        // 12 * 12 * 12 = 1728
        int volume = length * width * height;
        // Revert
        length -= PADDING; width -= PADDING; height -= PADDING;
        return volume;
    } catch (Throwable t) {
        MyExceptionReporter mer = new MyExceptionReporter();
        mer.report(t); // Sanitize
        return -1; // Non-positive error code
    }
}
```

94

94

## Quy tắc ERR03-J: Sửa lỗi

- Cách thức 1:

```
// ...
} catch (Throwable t) {
    MyExceptionReporter mer = new MyExceptionReporter();
    // Sanitize
    mer.report(t);
    // Revert
    length -= PADDING; width -= PADDING; height -= PADDING;
    return -1;
}
```

95

95

## Quy tắc ERR03-J: Sửa lỗi

- Cách thức 2: sử dụng mệnh đề finally

```
protected int getVolumePackage(int weight) {
    length += PADDING;
    width += PADDING;
    height += PADDING;
    try {
        if (length <= PADDING || width <= PADDING || height <= PADDING ||
            length > MAX_DIMENSION + PADDING ||
            width > MAX_DIMENSION + PADDING ||
            height > MAX_DIMENSION + PADDING ||
            weight <= 0 || weight > 20) {
            throw new IllegalArgumentException();
        }
        int volume = length * width * height; // 12 * 12 * 12 = 1728
        return volume;
    } catch (Throwable t) {
        MyExceptionReporter mer = new MyExceptionReporter();
        mer.report(t); // Sanitize
        return -1; // Non-positive error code
    } finally {
        // Revert
        length -= PADDING; width -= PADDING; height -= PADDING;
    }
}
```

96

96

## Quy tắc ERR03-J: Sửa lỗi

- Cách thức 3: Kiểm duyệt đầu vào

```
protected int getVolumePackage(int weight) {
    try {
        if (length <= 0 || width <= 0 || height <= 0 ||
            length > MAX_DIMENSION || width > MAX_DIMENSION ||
            height > MAX_DIMENSION ||
            weight <= 0 || weight > 20) {
            throw new IllegalArgumentException(); // Validate first
        }
    } catch (Throwable t) {
        MyExceptionReporter mer = new MyExceptionReporter();
        mer.report(t); // Sanitize
        return -1;
    }

    length += PADDING;
    width  += PADDING;
    height += PADDING;

    int volume = length * width * height;
    length -= PADDING; width -= PADDING; height -= PADDING;
    return volume;
}
```

97

97

## Quy tắc ERR03-J: Sửa lỗi

- Cách thức 4: Tránh thay đổi đối tượng

➤ Có thể khiến mã nguồn trở nên phức tạp và khó hiểu

```
protected int getVolumePackage(int weight) {
    try {
        if (length <= 0 || width <= 0 || height <= 0 ||
            length > MAX_DIMENSION || width > MAX_DIMENSION ||
            height > MAX_DIMENSION || weight <= 0 || weight > 20) {
            throw new IllegalArgumentException(); // Validate first
        }
    } catch (Throwable t) {
        MyExceptionReporter mer = new MyExceptionReporter();
        mer.report(t); // Sanitize
        return -1;
    }

    int volume = (length + PADDING) * (width + PADDING) *
                (height + PADDING);
    return volume;
}
```

98

98

## Quy tắc ERR04-J

- Không kết thúc trong mệnh đề `finally`

- Ví dụ lỗi:

```
private static boolean doLogic() {
    try {
        throw new IllegalStateException();
    } finally {
        System.out.println("logic done");
        return true;
    }
}
```

- Sửa lỗi:

```
private static boolean doLogic() {
    try {
        throw new IllegalStateException();
    } finally {
        System.out.println("logic done");
    }
    // Any return statements must go here;
    // applicable only when exception is thrown conditionally
}
```

99

99

## Quy tắc ERR05-J

- Không đưa ra bên ngoài ngoại lệ mà nó xảy ra trong mệnh đề `finally`

- Ví dụ lỗi:

```
public class Operation {
    public static void doOperation(String some_file) {
        // ... code to check or set character encoding ...
        try {
            BufferedReader reader =
                new BufferedReader(new FileReader(some_file));
            try {
                // Do operations
            } finally {
                reader.close();
                // ... Other cleanup code ...
            }
        } catch (IOException x) {
            // Forward to handler
        }
    }
}
```

100

100

## Quy tắc ERR05-J: Ví dụ - Sửa lỗi

- Cách thức 1

```
public class Operation {  
    public static void doOperation(String some_file) {  
        // ... code to check or set character encoding ...  
        try {  
            BufferedReader reader =  
                new BufferedReader(new FileReader(some_file));  
            try {  
                // Do operations  
            } finally {  
                try {  
                    reader.close();  
                } catch (IOException ie) {  
                    // Forward to handler  
                }  
                // ... Other clean-up code ...  
            }  
        } catch (IOException x) {  
            // Forward to handler  
        }  
    }  
}
```

101

101

## Quy tắc ERR05-J: Ví dụ - Sửa lỗi

- Cách thức 2: Sử dụng try-with-resource

```
public static void doOperation(String some_file) {  
    // ... code to check or set character encoding ...  
    try ( // try-with-resources  
        BufferedReader reader =  
            new BufferedReader(new FileReader(some_file))) {  
        // Do operations  
    } catch (IOException ex) {  
        System.err.println("thrown exception: " + ex.toString());  
        Throwable[] suppressed = ex.getSuppressed();  
        for (int i = 0; i < suppressed.length; i++) {  
            System.err.println("suppressed exception: "  
                + suppressed[i].toString());  
        }  
        // Forward to handler  
    }  
}
```

102

102

## Quy tắc ERR07-J

- Không tung ra ngoại lệ RuntimeException, Exception
  - Ngoại lệ được tung ra hoặc ủy nhiệm phải là cụ thể
- Ví dụ lỗi:

```
boolean isCapitalized(String s) {  
    if (s == null) {  
        throw new RuntimeException("Null String");  
    }  
    if (s.equals("")) {  
        return true;  
    }  
    String first = s.substring(0, 1);  
    String rest = s.substring(1);  
    return (first.equals(first.toUpperCase()) &&  
            rest.equals(rest.toLowerCase()));  
}
```

- Sửa lỗi:

```
if (s == null) {  
    throw new NullPointerException();  
}
```

103

103

## Quy tắc ERR08-J

- Chủ động kiểm tra con trỏ null, thay vì bắt ngoại lệ NullPointerException hoặc ngoại lệ cha của nó
  - Tương tự: ArrayIndexOutOfBoundsException, RuntimeException, Exception, hoặc Throwable
- Ví dụ lỗi:

```
boolean isName(String s) {  
    try {  
        String names[] = s.split(" ");  
        if (names.length != 2) {  
            return false;  
        }  
        return (isCapitalized(names[0]) && isCapitalized(names[1]));  
    } catch (NullPointerException e) {  
        return false;  
    }  
}
```

- Sửa lỗi:

```
boolean isName(String s) {  
    if (s == null) {  
        return false;  
    }  
    String names[] = s.split(" ");  
    if (names.length != 2) {  
        return false;  
    }  
    return (isCapitalized(names[0]) && isCapitalized(names[1]));  
}
```

104

104

## Quy tắc ERR08-J: Ví dụ lỗi khác(1)

```
public interface Log {  
    void write(String messageToLog);  
}  
  
public class FileLog implements Log {  
    private final FileWriter out;  
  
    FileLog(String logFileName) throws IOException {  
        out = new FileWriter(logFileName, true);  
    }  
  
    public void write(String messageToLog) {  
        // write message to file  
    }  
}  
  
public class ConsoleLog implements Log {  
    public void write(String messageToLog) {  
        System.out.println(messageToLog); // write message to console  
    }  
}
```

105

105

## Quy tắc ERR08-J: Ví dụ lỗi khác(2)

```
class Service {  
    private Log log;  
  
    Service() {  
        this.log = null; // no logger  
    }  
  
    Service(Log log) {  
        this.log = log; // set the specified logger  
    }  
  
    public void handle() {  
        try {  
            log.write("Request received and handled");  
        } catch (NullPointerException npe) {  
            // Ignore  
        }  
    }  
  
    public static void main(String[] args) throws IOException {  
        Service s = new Service(new FileLog("logfile.log"));  
        s.handle();  
  
        s = new Service(new ConsoleLog());  
        s.handle();  
    }  
}
```

106

106

## Quy tắc ERR08-J: Ví dụ khác – Sửa lỗi

```
public interface Log {  
    public static final Log NULL = new Log() {  
        public void write(String messageToLog) {  
            // do nothing  
        }  
    };  
    void write(String messageToLog);  
}  
  
class Service {  
    private final Log log = Log.NULL;  
    // ...  
}
```

107

107

## 8. CÁC QUY TẮC VÀO-RA DỮ LIỆU

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

108

108

## Quy tắc FIO00-J

- Không thao tác file đang ở trong thư mục chia sẻ
- Các HĐH cung cấp cơ chế file-locking để đảm bảo chỉ có một tiến trình được truy cập vào file.
- Vấn đề ATBM phát sinh với file trong thư mục chia sẻ: đối phương có thể sử dụng tiến trình truy cập trước trên file bị khóa và yêu cầu tiến trình nạn nhân truy cập vào sau  
→ khiến cho tiến trình nạn nhân bị treo

109

109

## Quy tắc FIO00-J: Ví dụ 1

```
String file = /* provided by user */;
InputStream in = null;
try {
    in = new FileInputStream(file);
    // ...
} finally {
    try {
        if (in != null) {
            in.close();
        }
    } catch (IOException x) {
        // handle error
    }
}
```

110

110

## Quy tắc FIO00-J: Ví dụ 2

- Sử dụng try-with-resource chỉ đảm bảo file được đóng khi có ngoại lệ

```
String filename = /* provided by user */;
Path path = new File(filename).toPath();
try (InputStream in = Files.newInputStream(path)) {
    // read file
} catch (IOException x) {
    // handle error
}
```

111

111

## Quy tắc FIO00-J: Ví dụ 3

- Sử dụng isRegularFile() nhưng vẫn gặp vấn đề với symbolic link

```
String filename = /* provided by user */;
Path path = new File(filename).toPath();
try {
    BasicFileAttributes attr =
        Files.readAttributes(path, BasicFileAttributes.class);

    // Check
    if (!attr.isRegularFile()) {
        System.out.println("Not a regular file");
        return;
    }
    // other necessary checks

    // Use
    try (InputStream in = Files.newInputStream(path)) {
        // read file
    }
} catch (IOException x) {
    // handle error
}
```

112

112

## Quy tắc FIO00-J: Ví dụ 4

- Kiểm tra thuộc tính NOFOLLOW\_LINKS để không cho phép sử dụng symbolic link nhưng vẫn gặp vấn đề TOCTOU

```
String filename = /* provided by user */;
Path path = new File(filename).toPath();
try {
    BasicFileAttributes attr = Files.readAttributes(
        path, BasicFileAttributes.class, LinkOption.NOFOLLOW_LINKS);

    // Check
    if (!attr.isRegularFile()) {
        System.out.println("Not a regular file");
        return;
    }
    // other necessary checks

    // Use
    try (InputStream in = Files.newInputStream(path)) {
        // read file
    };
} catch (IOException x) {
    // handle error
}
```

113

113

## Quy tắc FIO00-J: Ví dụ 5

- Thực hiện kỹ thuật check-use-check, nhưng vẫn chưa giải quyết được toàn bộ vấn đề

```
String filename = /* provided by user */;
Path path = new File(filename).toPath();
try {
    BasicFileAttributes attr = Files.readAttributes(
        path, BasicFileAttributes.class, LinkOption.NOFOLLOW_LINKS);
    Object fileKey = attr.fileKey();

    // Check
    if (!attr.isRegularFile()) {
        System.out.println("Not a regular file");
        return;
    }
    // other necessary checks
    // Use
    try (InputStream in = Files.newInputStream(path)) {

        // Check
        BasicFileAttributes attr2 = Files.readAttributes(
            path, BasicFileAttributes.class, LinkOption.NOFOLLOW_LINKS
        );
        Object fileKey2 = attr2.fileKey();
        if (fileKey != fileKey2) {
            System.out.println("File has been tampered with");
        }

        // read file
    };
} catch (IOException x) {
    // handle error
}
```

114

114

## Quy tắc FIO00-J: Sửa lỗi

- Sử dụng `isInSecureDir()` đảm bảo file và thư mục chứa nó thuộc sở hữu của người dùng hoặc tài khoản quản trị và file cùng thư mục này không cấp quyền ghi cho các tài khoản khác.

```
public static boolean isInSecureDir(Path file) {  
    return isInSecureDir(file, null);  
}  
public static boolean isInSecureDir(Path file, UserPrincipal user)  
{  
    return isInSecureDir(file, user, 5);  
}
```

115

115

## Quy tắc FIO00-J: Sửa lỗi(tiếp)

```
/**  
 * Indicates whether file lives in a secure directory relative  
 * to the program's user  
 * @param file Path to test  
 * @param user User to test. If null, defaults to current user  
 * @param symlinkDepth Number of symbolic links allowed  
 * @return true if file's directory is secure.  
 */  
public static boolean isInSecureDir(Path file, UserPrincipal user,  
                                    int symlinkDepth) {  
    if (!file.isAbsolute()) {  
        file = file.toAbsolutePath();  
    } if (symlinkDepth <= 0) {  
        // Too many levels of symbolic links  
        return false;  
    }  
}
```

116

116

## Quy tắc FIO00-J: Sửa lỗi(tiếp)

```
// Get UserPrincipal for specified user and superuser
FileSystem fileSystem =
    Paths.get(file.getRoot().toString()).getFileSystem();
UserPrincipalLookupService upls =
    fileSystem.getUserPrincipalLookupService();
UserPrincipal root = null;
try {
    root = upls.lookupPrincipalByName("root");
    if (user == null) {
        user = upls.lookupPrincipalByName(System.getProperty("user.name"));
    }
    if (root == null || user == null) {
        return false;
    }
} catch (IOException x) {
    return false;
}
```

117

117

## Quy tắc FIO00-J: Sửa lỗi(tiếp)

```
// If any parent dirs (from root on down) are not secure,
// dir is not secure
for (int i = 1; i <= file.getNameCount(); i++) {
    Path partialPath = Paths.get(file.getRoot().toString(),
        file.subpath(0, i).toString());

    try {
        if (Files.isSymbolicLink(partialPath)) {
            if (!isInSecureDir(Files.readSymbolicLink(partialPath),
                user, symlinkDepth - 1)) {
                // Symbolic link, linked-to dir not secure
                return false;
            }
        } else {
            UserPrincipal owner = Files.getOwner(partialPath);
            if (!user.equals(owner) && !root.equals(owner)) {
                // dir owned by someone else, not secure
                return false;
            }
        }
    }
```

118

118

## Quy tắc FIO00-J: Sửa lỗi(tiếp)

```
PosixFileAttributes attr =
    Files.readAttributes(partialPath, PosixFileAttributes.class);
Set<PosixFilePermission> perms = attr.permissions();
if (perms.contains(PosixFilePermission.GROUP_WRITE) ||
    perms.contains(PosixFilePermission.OTHERS_WRITE)) {
    // Someone else can write files, not secure
    return false;
}
} catch (IOException x) {
    return false;
}
}

return true;
}
```

119

119

## Quy tắc FIO00-J: Sửa lỗi(tiếp)

```
String filename = /* Provided by user */;
Path path = new File(filename).toPath();
try {
    if (!isInSecureDir(path)) {
        System.out.println("File not in secure directory");
        return;
    }

    BasicFileAttributes attr = Files.readAttributes(
        path, BasicFileAttributes.class, LinkOption.NOFOLLOW_LINKS);

    // Check
    if (!attr.isRegularFile()) {
        System.out.println("Not a regular file");
        return;
    }
    // Other necessary checks

    try (InputStream in = Files.newInputStream(path)) {
        // Read file
    }
} catch (IOException x) {
    // Handle error
}
```

120

120

## Quy tắc FIO01-J

- Tạo file với quyền truy cập phù hợp

- Ví dụ lỗi: 

```
Writer out = new FileWriter("file");
```

- Sửa lỗi:

```
Path file = new File("file").toPath();

// Throw exception rather than overwrite existing file
Set<OpenOption> options = new HashSet<OpenOption>();
options.add(StandardOpenOption.CREATE_NEW);
options.add(StandardOpenOption.APPEND);
// File permissions should be such that only user may read/write file
Set<PosixFilePermission> perms =
    PosixFilePermissions.fromString("rw-----");
FileAttribute<Set<PosixFilePermission>> attr =
    PosixFilePermissions.asFileAttribute(perms);

try (SeekableByteChannel sbc =
      Files.newByteChannel(file, options, attr)) {
    // write data
};
```

121

121

## Quy tắc FIO02-J

- Phát hiện và xử lý lỗi liên quan đến file

- Ví dụ lỗi 1: 

```
File file = new File(args[0]);
file.delete();
```

- Sửa lỗi:

```
File file = new File(args[0]);
if (!file.delete()) {
    System.out.println("Deletion failed");
}
```

- Sửa lỗi: sử dụng phương thức java.nio.file.Files.delete()

```
Path file = new File(args[0]).toPath();
try {
    Files.delete(file);
} catch (IOException x) {
    System.out.println("Deletion failed");
    // handle error
}
```

122

122

## Quy tắc FIO03-J

- Xóa file tạm trước khi kết thúc chương trình:
  - File tạm nên được tạo trong thư mục an toàn và sử dụng như quy tắc FIO00-J và FIO01-J
  - File tạm cần được xóa ngay cả khi JVM kết thúc bất thường
- Ví dụ lỗi:

123

123

## Quy tắc FIO03-J: Ví dụ lỗi

```
class TempFile {  
    public static void main(String[] args) throws IOException{  
        File f = File.createTempFile("tempnam",".tmp");  
        FileOutputStream fop = null;  
        try {  
            fop = new FileOutputStream(f);  
            String str = "Data";  
            fop.write(str.getBytes());  
            fop.flush();  
        } finally {  
            // Stream/file still open; file will  
            // not be deleted on Windows systems  
            // Delete the file when the JVM terminates  
            f.deleteOnExit();  
  
            if (fop != null) {  
                try {  
                    fop.close();  
                } catch (IOException x) {  
                    // handle error  
                }  
            }  
        }  
    }  
}
```

124

124

## Quy tắc FIO03-J: Sửa lỗi

```
class TempFile {  
    public static void main(String[] args) {  
        Path tempFile = null;  
        try {  
            tempFile = Files.createTempFile("tempnam", ".tmp");  
            try (BufferedWriter writer =  
                 Files.newBufferedWriter(tempFile, Charset.forName("UTF8"),  
                                         StandardOpenOption.DELETE_ON_CLOSE)) {  
                // write to file  
            }  
            System.out.println("Temporary file write done, file erased");  
        } catch (FileAlreadyExistsException x) {  
            System.err.println("File exists: " + tempFile);  
        } catch (IOException x) {  
            // Some other sort of failure, such as permissions.  
            System.err.println("Error creating temporary file: " + x);  
        }  
    }  
}
```

125

125

## Quy tắc FIO03-J: Sửa lỗi

- Khi không có thư mục an toàn cho file tạm, có thể sử dụng một số cơ chế thay thế:
  - Cơ chế IPC khác như socket hoặc RPC
  - JNI(Java Native Interface)
  - Ánh xạ bộ nhớ chính (memory-mapped files)
  - Chia sẻ dữ liệu trong bộ nhớ heap của JVM

126

126

## Quy tắc FIO04-J

- Đóng luồng vào ra khi không còn sử dụng tới
- Ví dụ lỗi:

```
public int processFile(String fileName)
    throws IOException, FileNotFoundException {
    FileInputStream stream = new FileInputStream(fileName);
    BufferedReader bufRead =
        new BufferedReader(new InputStreamReader(stream));
    String line;
    while ((line = bufRead.readLine()) != null) {
        sendLine(line);
    }
    return 1;
}
```

127

127

## Quy tắc FIO04-J: Ví dụ - Sửa lỗi

- Cách thức 1: Sử dụng mệnh đề finally

```
try {
    final FileInputStream stream = new FileInputStream(fileName);
    try {
        final BufferedReader bufRead =
            new BufferedReader(new InputStreamReader(stream));
        String line;
        while ((line = bufRead.readLine()) != null) {
            sendLine(line);
        }
    } finally {
        if (stream != null) {
            try {
                stream.close();
            } catch (IOException e) {
                // forward to handler
            }
        }
    }
} catch (IOException e) {
    // forward to handler
}
```

128

128

## Quy tắc FIO04-J: Ví dụ - Sửa lỗi

- Cách thức 2: Sử dụng try-with-resource

```
try (FileInputStream stream = new FileInputStream(fileName);
     BufferedReader bufRead =
         new BufferedReader(new InputStreamReader(stream))) {
    String line;
    while ((line = bufRead.readLine()) != null) {
        sendLine(line);
    }
} catch (IOException e) {
    // forward to handler
}
```

129

129

## Quy tắc FIO05-J

- Không bộ lộ bộ đệm sử dụng cho phương thức wrap() và duplicate() tới thành phần không tin cậy
  - Khi bộ đệm bị sửa thì dữ liệu đóng gói vào bộ đệm cũng bị sửa
- Ví dụ lỗi:

```
final class Wrap {
    private char[] dataArray;

    public Wrap() {
        dataArray = new char[10];
        // Initialize
    }

    public CharBuffer getBufferCopy() {
        return CharBuffer.wrap(dataArray);
    }
}
```

130

130

## Quy tắc FIO05-J: Ví dụ – Sửa lỗi

- Cách thức 1

```
public CharBuffer getBufferCopy() {  
    return CharBuffer.wrap(dataArray).asReadOnlyBuffer();  
}
```

- Cách thức 2

```
public CharBuffer getBufferCopy() {  
    CharBuffer cb = CharBuffer.allocate(dataArray.length);  
    cb.put(dataArray);  
    return cb;  
}
```

131

131

## Quy tắc FIO05-J – Ví dụ khác

```
final class Dup {  
    CharBuffer cb;  
  
    public Dup() {  
        cb = CharBuffer.allocate(10);  
        // Initialize  
    }  
  
    public CharBuffer getBufferCopy() {  
        return cb.duplicate();  
    }  
}
```

- Sửa lỗi

```
public CharBuffer getBufferCopy() {  
    return cb.asReadOnlyBuffer();  
}
```

132

132

## Quy tắc FIO06-J

- Mỗi luồng vào ra chỉ sử dụng một bộ đệm
- Ví dụ lỗi:

```
public final class InputLibrary {  
    public static char getChar() throws EOFException, IOException {  
        // wrapper  
        BufferedInputStream in = new BufferedInputStream(System.in);  
        int input = in.read();  
        if (input == -1) {  
            throw new EOFException();  
        }  
        // Down casting is permitted because InputStream  
        // guarantees read() in range  
        // 0..255 if it is not -1  
        return (char) input;  
    }  
}
```

133

133

## Quy tắc FIO06-J – Sửa lỗi

```
public final class InputLibrary {  
    private static BufferedInputStream in =  
        new BufferedInputStream(System.in);  
  
    public static char getChar() throws EOFException, IOException {  
        int input = in.read();  
        if (input == -1) {  
            throw new EOFException();  
        }  
        in.skip(1); // This statement is to advance to the next line  
                   // The noncompliant code example deceptively  
                   // appeared to work without it (in some cases)  
        return (char) input;  
    }  
}
```

134

134

## Quy tắc FIO08-J

- Sử dụng biến kiểu int để nhận giá trị trả về của phương thức đọc 1 byte hoặc 1 ký tự

- Ví dụ lỗi:

```
FileInputStream in;
// initialize stream
byte data;
while ((data = (byte) in.read()) != -1) {
    // ...
}
```

- Sửa lỗi:

```
FileInputStream in;
// initialize stream
int inbuff;
byte data;
while ((inbuff = in.read()) != -1) {
    data = (byte) inbuff;
    // ...
}
```

135

135

## Quy tắc FIO09-J

- Phương thức write() chỉ ghi các giá trị số nguyên 0..255

- Ví dụ lỗi:

```
class ConsoleWrite {
    public static void main(String[] args) {
        // Any input value > 255 will result in unexpected output
        System.out.write(Integer.valueOf(args[0]));
        System.out.flush();
    }
}
```

- Sửa lỗi:

```
class FileWrite {
    public static void main(String[] args)
        throws NumberFormatException, IOException {
        // Perform range checking
        int value = Integer.valueOf(args[0]);
        if (value < 0 || value > 255) {
            throw new ArithmeticException("Value is out of range");
        }

        System.out.write(value);
        System.out.flush();
    }
}
```

136

136

## Quy tắc FIO010-J

- Kiểm tra mảng có được lấp đầy khi sử dụng phương thức `read()` đọc dữ liệu và đưa vào mảng
- Ví dụ lỗi:

```
public static String readBytes(InputStream in) throws IOException {  
    byte[] data = new byte[1024];  
    if (in.read(data) == -1) {  
        throw new EOFException();  
    }  
    return new String(data, "UTF-8");  
}
```

137

137

## Quy tắc FIO010-J: Ví dụ - Sửa lỗi

```
public static String readBytes(InputStream in) throws IOException {  
    int offset = 0;  
    int bytesRead = 0;  
    byte[] data = new byte[1024];  
    while ((bytesRead = in.read(data, offset, data.length - offset))  
        != -1) {  
        offset += bytesRead;  
        if (offset >= data.length) {  
            break;  
        }  
    }  
    String str = new String(data, "UTF-8");  
    return str;  
}
```

- Lưu ý: nếu không bắt buộc phải lấp đầy mảng thì phải xác định số byte đã đọc được và chỉ xử lý trên số byte này.

138

138

## Quy tắc FIO11-J

- Không đọc dữ liệu nhị phân như là dữ liệu ký tự

139

139

## Quy tắc FIO14-J

- Đảm bảo các luồng vào ra được đóng khi chương trình kết thúc
- Ví dụ lỗi:

```
public class InterceptExit {  
    public static void main(String[] args)  
        throws FileNotFoundException {  
        InputStream in = null;  
        try {  
            in = new FileInputStream("file");  
            System.out.println("Regular code block");  
            // Abrupt exit such as ctrl + c key pressed  
            System.out.println("This never executes");  
        } finally {  
            if (in != null) {  
                try {  
                    in.close(); // this never executes either  
                } catch (IOException x) {  
                    // handle error  
                }  
            }  
        }  
    }  
}
```

140

140

## Quy tắc FIO14-J: Ví dụ - Sửa lỗi

- Hook vào quá trình shutdown
- Lưu ý: Chỉ sử dụng 1 luồng để hook

```
public class Hook {  
    public static void main(String[] args) {  
        try {  
            final InputStream in = new FileInputStream("file");  
            Runtime.getRuntime().addShutdownHook(new Thread() {  
                public void run() {  
                    // Log shutdown and close all resources  
                    in.close();  
                }  
            });  
  
            // ...  
        } catch (IOException x) {  
            // handle error  
        } catch (FileNotFoundException x) {  
            // handle error  
        }  
    }  
}
```

141

141

## Quy tắc FIO16-J

- Chuẩn hóa đường dẫn file trước khi kiểm duyệt để tránh lỗ hổng duyệt thư mục hoặc truy cập vượt phạm vi.
- Ví dụ lỗi: kiểm soát không cho người dùng truy cập ngoài thư mục của họ. Tuy nhiên, mã nguồn không kiểm soát họ có thể ra ngoài bằng shortcut/symbolic link

```
public static void main(String[] args) {  
    File f = new File(System.getProperty("user.home") +  
        System.getProperty("file.separator") + args[0]);  
  
    String absPath = f.getAbsolutePath();  
  
    if (!isInSecureDir(Paths.get(absPath))) {  
        throw new IllegalArgumentException();  
    }  
    if (!validate(absPath)) { // Validation  
        throw new IllegalArgumentException();  
    }  
}
```

142

142

## Quy tắc FIO16-J: Ví dụ - Sửa lỗi

```
public static void main(String[] args) throws IOException {
    File f = new File(System.getProperty("user.home") +
        System.getProperty("file.separator") + args[0]);
    String canonicalPath = f.getCanonicalPath();

    if (!isInSecureDir(Paths.get(canonicalPath))) {
        throw new IllegalArgumentException();
    }
    if (!validate(canonicalPath)) { // Validation
        throw new IllegalArgumentException();
    }
}
```

143

143

## 9. CÁC QUY TẮC VỀ NỀN TẢNG

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

144

144

## Quy tắc SEC00-J

- Không cho phép các khối lệnh đặc quyền để lộ dữ liệu nhạy cảm qua ranh giới tin cậy

- Ví dụ lỗi

```
public static FileInputStream openPasswordFile()
    throws FileNotFoundException {
    final String password_file = "password";
    FileInputStream fin = null;
    try {
        fin = AccessController.doPrivileged(
            new PrivilegedExceptionAction<FileInputStream>() {
                public FileInputStream run() throws FileNotFoundException {
                    // Sensitive action; can't be done outside privileged block
                    FileInputStream in = new FileInputStream(password_file);
                    return in;
                }
            });
    } catch (PrivilegedActionException x) {
        Exception cause = x.getException();
        if (cause instanceof FileNotFoundException) {
            throw (FileNotFoundException) cause;
        } else {
            throw new Error("Unexpected exception type", cause);
        }
    }
    return fin;
}
```

145

145

## Quy tắc SEC00-J: Ví dụ - Sửa lỗi

- Sửa lỗi 1:

```
private static FileInputStream openPasswordFile()
    throws FileNotFoundException {
    // ...
}
```

- Sửa lỗi 2: Ẩn thông tin chi tiết của ngoại lệ

```
class PasswordManager {

    public static void changePassword() {
        FileInputStream fin = openPasswordFile();
        if (fin == null) {
            // no password file; handle error
        }

        // test old password with password in file contents; change password
    }
}
```

146

146

## Quy tắc SEC00-J: Sửa lỗi 2(tiếp)

```
private static FileInputStream openPasswordFile() {
    final String password_file = "password";
    final FileInputStream fin[] = { null };
    AccessController.doPrivileged(new PrivilegedAction<Void>() {
        public Void run() {
            try {
                // Sensitive action; can't be done outside
                // doPrivileged() block
                fin[0] = new FileInputStream(password_file);
            } catch (FileNotFoundException x) {
                // report to handler
            }
            return null;
        }
    });
    return fin[0];
}
```

147

147

## Quy tắc SEC02-J

- Không thực hiện kiểm tra ATBM dựa trên nguồn không đáng tin cậy
- Ví dụ lỗi: Java 1.5 không định nghĩa `final` cho lớp File dẫn đến lỗ hổng TOCTOU

```
public RandomAccessFile openFile(final java.io.File f) {
    askUserPermission(f.getPath());
    // ...
    return (RandomAccessFile) AccessController.doPrivileged() {
        public Object run() {
            return new RandomAccessFile(f.getPath());
        }
    }
}
```

Tấn công:

```
public class BadFile extends java.io.File {
    private int count;
    public String getPath() {
        return (++count == 1) ? "/tmp/foo" : "/etc/passwd";
    }
}
```

148

148

## Quy tắc SEC02-J: Ví dụ - Sửa lỗi

- Cách thức 1: Sử dụng khai báo `final`
- Cách thức 2:

```
public RandomAccessFile openFile(java.io.File f) {  
    final java.io.File copy = new java.io.File(f.getPath());  
    askUserPermission(copy.getPath());  
    // ...  
    return (RandomAccessFile) AccessController.doPrivileged() {  
        public Object run() {  
            return new RandomAccessFile(copy.getPath());  
        }  
    }  
}
```

149

149

## Quy tắc SEC04-J

- Kiểm tra bảo mật để bảo vệ các phương thức nhạy cảm
- Ví dụ lỗi:

```
class SensitiveHash {  
    Hashtable<Integer, String> ht = new Hashtable<Integer, String>();  
  
    public void removeEntry(Object key) {  
        ht.remove(key);  
    }  
}
```

- Sửa lỗi

```
void removeEntry(Object key) {  
    check("removeKeyPermission");  
    ht.remove(key);  
}  
  
private void check(String directive) {  
    SecurityManager sm = System.getSecurityManager();  
    if (sm != null) {  
        sm.checkSecurityAccess(directive);  
    }  
}
```

150

150

## Quy tắc SEC04-J: Ví dụ khác

- Mã nguồn sau không đủ để thực thi chính sách kiểm soát truy cập chi tiết, ví dụ chỉ cho đọc các file .dtd

```
SecurityManager sm = System.getSecurityManager();  
  
if (sm != null) { // check whether file may be read  
    sm.checkRead("/local/schema.dtd");  
}
```

- Sửa lỗi: tự định nghĩa DTDPPermission mô tả chi tiết chính sách kiểm soát truy cập

```
SecurityManager sm = System.getSecurityManager();  
  
if (sm != null) { //check whether file can be read or not  
    DTDPPermission perm = new DTDPPermission("/local/", "readDTD");  
    sm.checkPermission(perm);  
}
```

151

151

## Quy tắc SEC06-J

- Không phụ thuộc việc xác thực chữ ký số cung cấp bởi URLClassLoader và java.util.jar
- Khi ứng dụng sử dụng thư viện hoặc plugin từ bên thứ 3 thông qua URL ClassLoader, cần phải kiểm tra chữ ký số để xác thực sự tin cậy
- Lớp URLClassLoader cùng các lớp con hoặc java.util.jar cung cấp cơ chế tự động xác thực chữ ký nhưng không kiểm tra sự tin cậy của khóa công khai → lỗi hỏng xác thực
- Cách thức 1: Kiểm tra thủ công

```
jarsigner -verify signed-updates-jar-file.jar
```

152

152

## Quy tắc SEC06-J: Xác thực chữ ký

- Kết thừa lớp URLClassLoader và ghi đè phương thức invokeClass()

```
public void invokeClass(String name, String[] args)
    throws ClassNotFoundException, NoSuchMethodException,
           InvocationTargetException, GeneralSecurityException,
           IOException {
    Class c = loadClass(name);
    Certificate[] certs =
        c.getProtectionDomain().getCodeSource().getCertificates();
    if (certs == null) {
        // return, do not execute if unsigned
        System.out.println("No signature!");
        return;
    }

    KeyStore ks = KeyStore.getInstance("JKS");
    ks.load(new FileInputStream(System.getProperty(
        "user.home"+ File.separator + "keystore.jks")),
        "loadkeystorepassword".toCharArray());
    // user is the alias
    Certificate pubCert = ks.getCertificate("user");
    // check with the trusted public key, else throws exception
    certs[0].verify(pubCert.getPublicKey());
}
```

153

153

## 9. CÁC QUY TẮC VỀ MÔI TRƯỜNG THỰC THI

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

154

154

## Quy tắc ENV00-J

- Không ký lên mã nguồn mà nó chỉ thực hiện các thao tác không yêu cầu đặc quyền

155

155

## Quy tắc ENV01-J

- Đóng gói tất cả các mã nguồn nhạy cảm vào một file jar duy nhất, ký và niêm phong nó.

156

156

## Quy tắc ENV02-J

- Không tin cậy vào giá trị từ các biến môi trường
- Ví dụ lỗi:

```
String username = System.getenv("USER");
```

- Sửa lỗi:

```
String username = System.getProperty("user.name");
```

157

157

## Quy tắc ENV04-J

- Không disable xác thực mã bytecode
- Ví dụ lỗi:

```
java -Xverify:none ApplicationName
```

- Sửa lỗi:

```
java -Xverify:all ApplicationName
```

158

158

## 10. CÁC QUY TẮC KHÁC

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

159

159

## Các quy tắc khác

- MSC00-J: Nếu có thể sử dụng SSL Socket thay cho TCP Socket
- MSC01-J: Không sử dụng vòng lặp rỗng vô tận
- MSC02-J: Sử dụng bộ sinh số ngẫu nhiên mật mã
- MSC03-J: Không đặt thông tin nhạy cảm trong mã nguồn

➤ Ví dụ:

```
class IPAddress {  
    String ipAddress = new String("172.16.254.1");  
    public static void main(String[] args) {  
        // ...  
    }  
}
```

➤ Sửa lỗi: Lưu trữ trên file cấu hình đặt trong thư mục an toàn

160

160

## Quy tắc MSC04-J

- Không gây thất thoát bộ nhớ
- Ví dụ lỗi:

```
public class Leak {  
    static Vector vector = new Vector();  
  
    public void useVector(int count) {  
        for (int n = 0; n < count; n++) {  
            vector.add(Integer.toString(n));  
        }  
        // ...  
        for (int n = count - 1; n > 0; n--) { // Free the memory  
            vector.removeElementAt(n);  
        }  
    }  
}
```

- Sửa lỗi: `vector.clear(); // Clear the vector`

161

161

## Quy tắc MSC05-J: Ví dụ 2

```
public class Storer {  
    private HashMap<Integer, String> hm = new HashMap<Integer, String>();  
  
    private void doSomething() {  
        // hm is used only here and never referenced again  
        hm.put(1, "java");  
        // ...  
    }  
}
```

- Sửa lỗi

```
public class Storer {  
    private void doSomething() {  
        HashMap<Integer, String> hm = new HashMap<Integer, String>();  
        hm.put(1, "java");  
        // ...  
    }  
}
```

162

162

## Quy tắc MSC05-J: Ví dụ 3

```
public class Stack {  
    private Object[] elements;  
    private int size = 0;  
    public Stack(int initialCapacity) {  
        this.elements = new Object[initialCapacity];  
    }  
  
    public void push(Object e) {  
        ensureCapacity();  
        elements[size++] = e;  
    }  
  
    public Object pop() { // This method causes memory leaks  
        if (size == 0) {  
            throw new EmptyStackException();  
        }  
        return elements[--size];  
    }  
  
    /*  
     * Ensure space for at least one more element, roughly  
     * doubling the capacity each time the array needs to grow.  
     */  
    private void ensureCapacity() {  
        if (elements.length == size) {  
            Object[] oldElements = elements;  
            elements = new Object[2 * elements.length + 1];  
            System.arraycopy(oldElements, 0, elements, 0, size);  
        }  
    }  
}
```

163

163

## Quy tắc MSC05-J: Ví dụ 3 – Sửa lỗi

```
public Object pop() {  
    if (size == 0) {  
        throw new EmptyStackException(); // Ensures object consistency  
    }  
    Object result = elements[--size];  
    elements[size] = null; // Eliminate obsolete reference  
    return result;  
}
```

164

164

## Quy tắc MSC05-J

- Không làm cạn kiệt bộ nhớ heap
- Ví dụ lỗi:

```
class ReadNames {  
    private Vector<String> names = new Vector<String>();  
    private final InputStreamReader input;  
    private final BufferedReader reader;  
  
    public ReadNames(String filename) throws IOException {  
        this.input = new FileReader(filename);  
        this.reader = new BufferedReader(input);  
    }  
  
    public void addNames() throws IOException {  
        try {  
            String newName;  
            while (((newName = reader.readLine()) != null) &&  
                  !(newName.equalsIgnoreCase("quit"))){  
                names.addElement(newName);  
                System.out.println("adding " + newName);  
            }  
        } finally {  
            input.close();  
        }  
    }  
  
    public static void main(String[] args) throws IOException {  
        if (args.length != 1) {  
            System.out.println("Arguments: [filename]");  
            return;  
        }  
        ShowHeapError demo = new ShowHeapError(args[0]);  
        demo.addNames();  
    }  
}
```

165

165

## Quy tắc MSC05-J: Ví dụ - Sửa lỗi

- Giới hạn kích thước file

```
class ReadNames {  
    public static final int fileSizeLimit = 1000000;  
  
    public ReadNames(String filename) throws IOException {  
        if (Files.size(Paths.get(filename)) > fileSizeLimit) {  
            throw new IOException("File too large");  
        }  
        this.input = new FileReader(filename);  
        this.reader = new BufferedReader(input);  
    }  
}
```

166

166

## Quy tắc MSC06-J

- Không thay đổi nội dung của Collection khi đang duyệt bằng Iterator

- Ví dụ lỗi:

```
List<String> list = new ArrayList<String>();
list.add("one");
list.add("two");

Iterator iter = list.iterator();
while (iter.hasNext()) {
    String s = (String)iter.next();
    if (s.equals("one")) {
        list.remove(s);
    }
}
```

- Sửa lỗi:

```
// ...
if (s.equals("one")) {
    iter.remove();
}
// ...
```

167

167

## 11. CHỐNG DỊCH NGƯỢC

Bùi Trọng Tùng,  
Trường Công nghệ thông tin và Truyền thông,  
Đại học Bách khoa Hà Nội

168

168

## Dịch ngược

- Là kỹ thuật để chuyển ngược từ mã bytecode (.class) về mã nguồn Java(.java)
- Khiến cho lô thông tin nhạy cảm có trong mã nguồn
- Các kỹ thuật chống dịch ngược:
  - Làm rối luồng xử lý, tham số của phương thức, tham chiếu..
  - Mã hóa xâu, hằng số nguyên
  - Mã hóa mã bytecode
  - Sử dụng ClassLoader
- Một số công cụ:
  - Miễn phí: ProGuard
  - Trả phí: Zelix KlassMaster

169