

# 표준 예외를 사용하라

## **표준 예외란?**

**자바 API에서 제공하는 예외로 대표적으로  
IllegalArgumentException, IllegalStateException 등이 있다.**

# **표준 예외를 왜 사용해야 할까?**

- 나의 API가 다른 사람이 익히고 사용하기 쉬워진다.
- 가독성이 향상되어 프로그램이 읽기 쉬워진다.
- 예외 클래스 수가 적을수록 메모리 사용량도 줄고 클래스를 적재하는 시간도 적게 걸린다.

# 표준 예외를 왜 사용해야 할까?

```
public class RuntimeException extends Exception {
    @java.io.Serial
    static final long serialVersionUID = -7034897190745766939L;

    public RuntimeException(String message) {
        super(message);
    }
    ...
}
```



```
public class Exception extends Throwable {
    @java.io.Serial
    static final long serialVersionUID = -3387516993124229948L;

    public Exception(String message) {
        super(message);
    }
    ...
}
```

# 표준 예외를 왜 사용해야 할까?

```
public class Throwable implements Serializable {  
    /** use serialVersionUID from JDK 1.0.2 for interoperability */  
    @java.io.Serial  
    private static final long serialVersionUID = -3042686055658047285L;  
  
    ...  
    public Throwable(String message) {  
        fillInStackTrace()  
        detailMessage = message;  
    }  
    ...  
}
```

## 표준 예외를 왜 사용해야 할까?

```
public synchronized Throwable fillInStackTrace() {  
    if (stackTrace != null ||  
        backtrace != null /* Out of protocol state */) {  
        fillInStackTrace( dummy: 0);  
        stackTrace = UNASSIGNED_STACK;  
    }  
    return this;  
}  
  
private native Throwable fillInStackTrace(int dummy);
```

- fillInStackTrace() 메서드는 실제 스택 트레이스를 채우는 로직을 담당한다.
- native 키워드가 붙어 있으므로, 이 메서드의 구현은 자바가 아닌 네이티브 코드(대부분 C나 C++)로 되어 있으며, JVM의 내부 구현에 속해 있다.

# 표준 예외를 왜 사용해야 할까?

예외 발생 시에는 JVM이 예외 객체를 생성하고, 스택 트레이스를 수집하며, 이 모든 작업은 비용이 발생한다.

- 예외 클래스의 수가 적고 그 구조가 간단할수록 이 과정에서의 성능 부하가 줄어듭니다.

# 표준 예외를 왜 사용해야 할까?

```
public class Throwable implements Serializable {  
    /** use serialVersionUID from JDK 1.0.2 for interoperability */  
    @java.io.Serial  
    private static final long serialVersionUID = -3042686055658047285L;  
  
    ...  
    public Throwable(String message) {  
        fillInStackTrace()  
        detailMessage = message;  
    }  
    ...  
}
```

예외는 직렬화할 수 있으며 이 사실만으로도 커스텀 예외를 만들지 않아야 할 근거로 충분하다.



## 널리 사용되는 표준 예외

예외	주요 쓰임
IllegalArgumentException	허용하지 않는 값이 인수로 건네졌을 때 (null은 NPE로 처리)
IllegalStateException	객체가 메서드를 수행하기 적절하지 않은 상태일 때
NullPointerException	null을 허용하지 않는 메서드에 null을 건넸을 때
IndexOutOfBoundsException	인덱스가 범위를 넘어섰을 때
ConcurrentModificationException	허용하지 않는 동시 수정이 발견됐을 때
UnsupportedOperationException	호출한 메서드를 지원하지 않을 때

# IllegalArgumentException



```
public void setAge(int age) {  
    if (age < 0) {  
        throw new IllegalArgumentException("나이는 음수가 될 수 없습니다.")  
    }  
    this.age = age;  
}
```

호출자가 인수로 부적절한 값을 넘길 때 던지는 예외로 나이를 할당하는 메서드에 음수가 할당되는 경우

# IllegalStateException




```
public void start() {  
    if (this.player == null) {  
        throw new IllegalStateException("player가 준비되지 않았습니다.");  
    }  
}
```

대상 객체의 상태가 호출된 메서드를 수행하기에 적절하지 않을 때 발생시킬 수 있는 예외이다.

- 게임에 사용자가 준비되지 않았을 때와 같은 경우

# NullPointerException




```
public void updateAge(Integer age) {  
    if (age == null) {  
        throw new NullPointerException("나이가 존재하지 않습니다.");  
    }  
    if (age < 0) {  
        throw new IllegalArgumentException("나이는 음수가 될 수 없습니다.");  
    }  
    this.age = age;  
}
```

**null 값을 허용하지 않는 메서드에 null을 건넬 때 발생시킬 수 있는 예외이다.**

- **IllegalArgumentException를 발생시킬 수도 있지만, 관례상 null을 건네면 NullPointerException을 발생시킨다.**

# UnsupportedOperationException



```
public static void main(String[] args) {  
    List<String> readOnlyElement = List.of("readOnly element");  
    readOnlyElement.remove(0);  
}
```

```
Exception in thread "main" java.lang.UnsupportedOperationException Create breakpoint  
    at java.base/java.util.ImmutableCollections.uoe(ImmutableCollections.java:142)  
    at java.base/java.util.ImmutableCollections$AbstractImmutableList.remove(ImmutableCollections.java:258)  
    at attraction.yong.exam.Main.main(Main.java:26)
```

클라이언트가 요청한 동작을 대상 객체가 지원하지 않을 때 발생하는 예외이다.

- 예를 들어 add나 remove를 허용하지 않는 List 구현체에 해당 요청을 보내는 경우 발생한다.

## 그외 표준 예외들

복소수나 유리수를 다루는 객체를 작성한다면

- `ArithmeticException`, `NumberFormatException`

단일 스레드 환경에서 적합한 동작을, 멀티 스레드에서 동작하려고 할 때

- 동시에 수정을 하는 상황을 확실히 검출할 수 있는 방법은 없기 때문에  
`ConcurrentModificationException`은 문제가 생길 가능성을 알려주는 역할을 한다.

# IllegalArgumentException VS IllegalStateException

인수 값이 무엇이었던 어차피 실패했을 거라면?


- `IllegalStateException`

그렇지 않으면

- `IllegalArgumentException`



# 커스텀 예외는?



```
public class UserNotFoundException extends RuntimeException {  
    public UserNotFoundException(String id) {  
        super("user(" + id + ")" + " is not found.")  
    }  
}
```

- 커스텀 예외는 예외 이름 자체가 정보를 전달할 수 있다. NoSuchElementException 보다 는 UserNotFoundException이 더 명확하다.
- 상세하게 예외 정보를 제공할 수 있고, 예외에 필요한 메시지, 전달할 정보의 데이터 등등을 한 곳에서 관리가 가능하다.
- 결론: 적절하게 사용하자(디렉토리와 클래스 관리, JVM 성능 문제등)



