예외, 예외처리, 예외전가 및 사용자 예외클래스

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

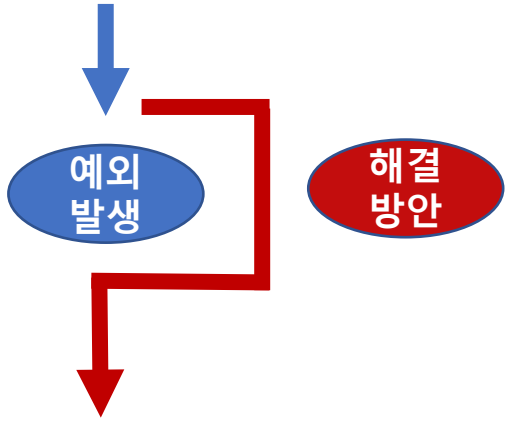
1 ☞ 예외(Exception)와 에러(Error)의 차이점

- **예외(Exception)**: 연산오류, 포맷오류 등 상황에 따라 개발자가 해결 가능한 오류

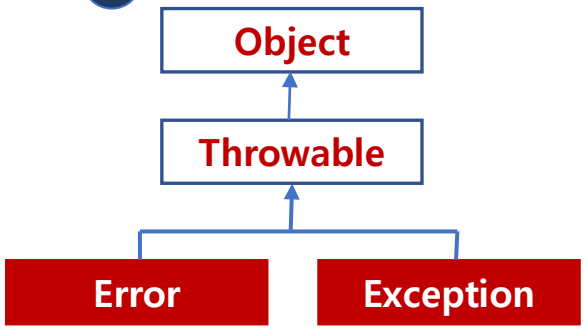
3

ex. a=1/0 → 연산불가

→ **해결책#1**: 최대값부여, **해결책#2**: 해당연산 제외

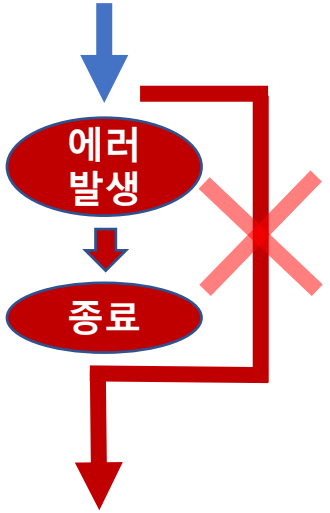


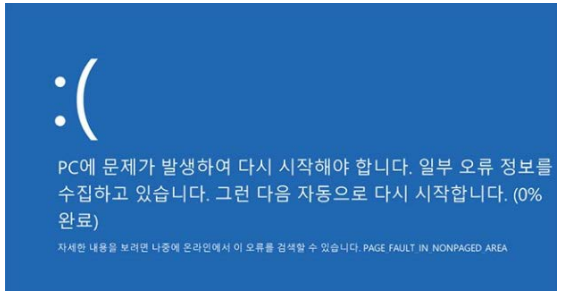
2



4 - **에러(Error)**: JVM 자체의 오류로 개발자가 해결할 수 없는 오류

ex. OutOfMemoryError, ThreadDeath
cf. windows의 파란화면

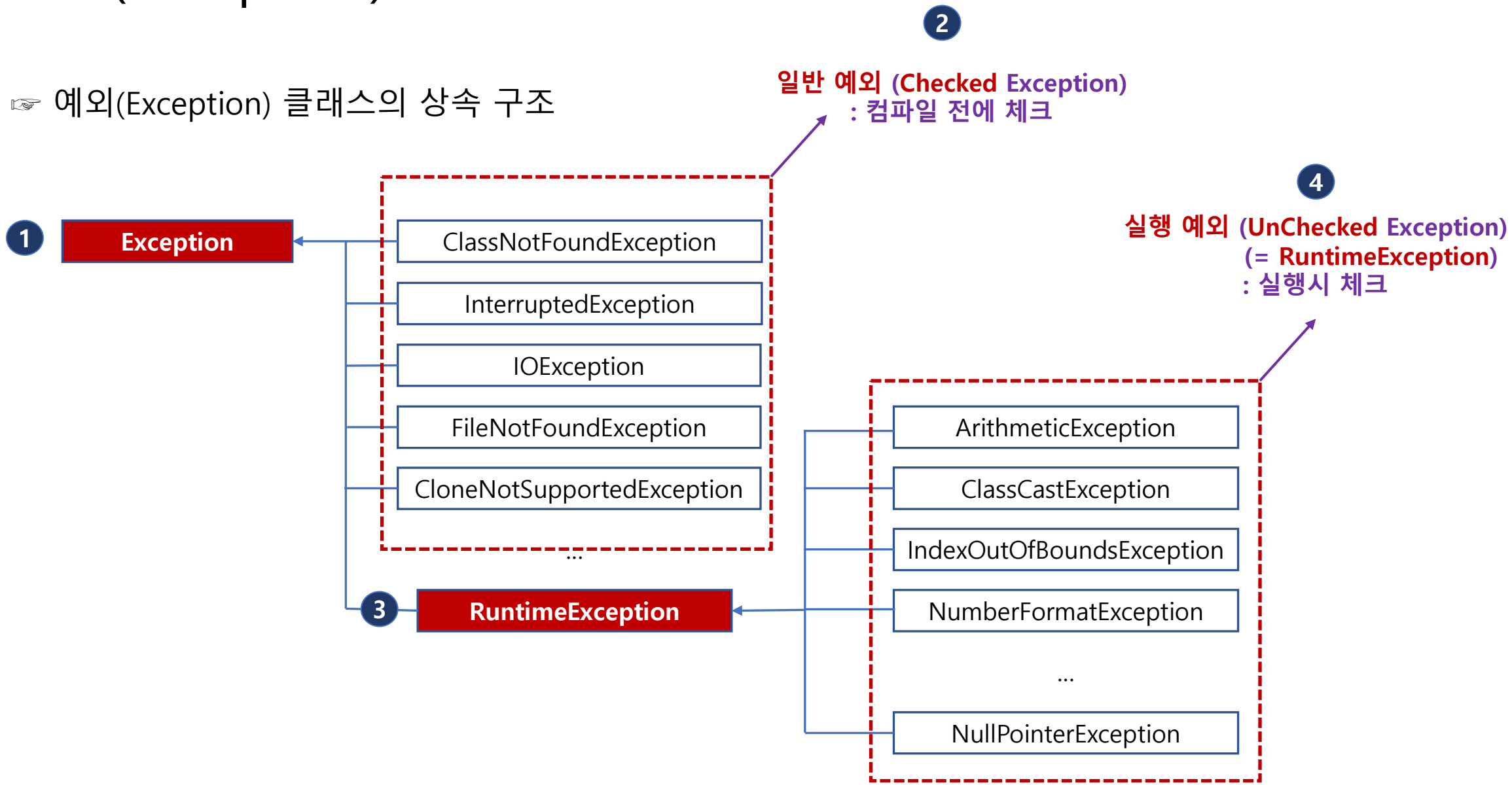




Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

☞ 예외(Exception) 클래스의 상속 구조



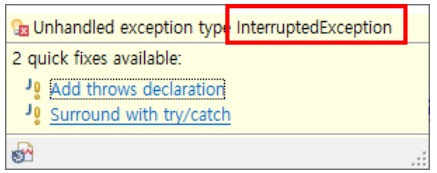
Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

예외처리를 하지 않으면 컴파일 자체가 불가능

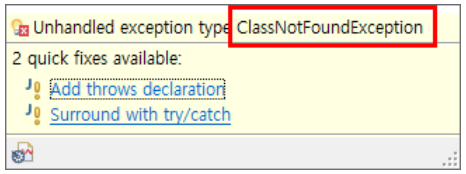
1 ☞ 일반 예외 (Checked Exception) → Exception으로 부터 바로 상속

2
```
public class A {
  public static void main(String[] ar){
    Thread.sleep(1000);
  }
}
```



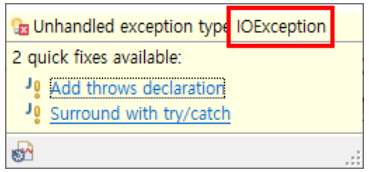
Thread 실행 중 interrupt 발생가능
InterruptedException 예외처리 필요

3
```
public class A {
  public static void main(String[] ar){
    Class cls  = Class.forName("java.lang.Object");
  }
}
```



Class가 없는 경우 예외 발생가능
ClassNotFoundException 예외처리 필요

4
```
public class A {
  public static void main(String[] ar){
    InputStreamReader isr = new InputStreamReader(System.in);
    isr.read();
 }
}
```



입출력 수행시 예외 발생가능
IOException 예외처리 필요

Copyright 2020. Donghyung Kim, All rights reserved

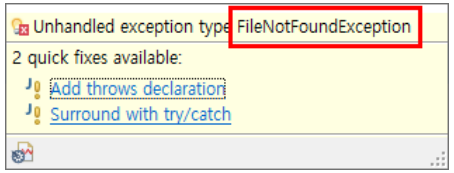
예외(Exception) 및 예외 처리

예외처리를 하지 않으면 컴파일 자체가 불가능

☞ 일반 예외 (Checked Exception) → Exception으로 부터 바로 상속

1

```
public class A {
  public static void main(String[] ar){
    FileInputStream fis = new FileInputStream("text.txt");
  }
}
```



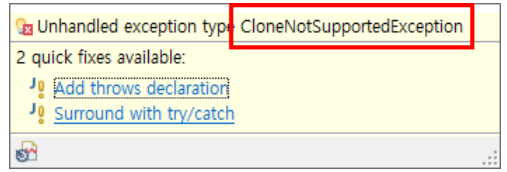
File이 없는 경우 예외 발생가능
FileNotFoundException 예외처리 필요

2

```
class B {
  @Override
  protected Object clone() throws CloneNotSupportedException {
    return super.clone();
  }
}
public class A {
  public static void main(String[] ar){
    B b1 = new B();
    B b2 = (B)b1.clone();
  }
}
```

3

참고. A 클래스 내부에서 clone을 호출하기 위해서는 B클래스가 clone 메서드를 오버라이딩 해야함
(A클래스 입장에서는 Object의 clone에 억세스 할 수 없기 때문



Class B가 Clonable 인터페이스를
구현하지 않은 경우 예외발생 가능
CloneNotSupportedException 발생

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

예외처리를 하지 않아도 컴파일은 가능
실행중 예외가 발생하면 프로그램 종료

1

☞ 실행예외 (RuntimeException) → RuntimeException으로 부터 바로 상속

2

```
public class A {
  public static void main(String[] ar){
    System.out.println(3/0);
  }
}
```

분모가 0인 연산 불가로 인한 예외 발생
ArithmeticException 발생

```
Exception in thread "main" java.lang.ArithmeticException:
        at pack01_exceptionhandling.EX02_UnCheckedExcepti
```

3

```
class A{}
class B extends A{}
public class Test {
  public static void main(String[] ar){
    A a = new A();
    B b = (B)a;
  }
}
```

Class 캐스팅이 불가능 한 경우 예외 발생가능
ClassCastException 발생

```
Exception in thread "main" java.lang.ClassCastException:
        at pack01_exceptionhandling.EX02_UnCheckedExcept
```

4

```
public class A {
  public static void main(String[] ar){
    int[] a = {1, 2, 3};
    System.out.println(a[3]);
  }
}
```

입출력 수행시 예외 발생가능
ArrayIndexOutOfBoundException 발생

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
        at pack01_exceptionhandling.EX02_UnCheckedException.EX02_UnC
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

예외처리를 하지 않아도 컴파일은 가능
실행중 예외가 발생하면 프로그램 종료

☞ 실행예외 (RuntimeException) → RuntimeException으로 부터 바로 상속

1

```
public class A {
  public static void main(String[] ar){
    int num = Integer.parseInt("10!");
  }
}
```

Number(숫자)가 아닌 것을 숫자로 바꿀 때 예외 발생
NumberFormatException 발생

```
Exception in thread "main" java.lang.NumberFormatException:
        at java.lang.NumberFormatException.forInputString(N
```

2

```
public class A {
  public static void main(String[] ar){
    String a = null;
    System.out.println(a.charAt(2));
  }
}
```

객체를 생성하지 않고 멤버를 사용할때 예외 발생가능
NullPointerException 발생

```
Exception in thread "main" java.lang.NullPointerException
        at pack01_exceptionhandling.EX02_UnCheckedExceptio
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

☞ 예외 처리

1 일반예외(Checked Exception): 컴파일 자체가 불가능
실행예외: 예외발생시 프로그램 종료 (예외메시지 출력)

예외처리 → 정상실행 가능

☞ 예외 처리 문법

2
```
try{
  //예외 발생 가능 코드
  //일반예외/실행예외 발생 가능 코드
}
catch(예외클래스이름 참조변수명){
  //해당예외가 발생한 경우 처리블록
}
finally{
  //예외 발생여부에 상관없이 무조건 실행블록
}
```

finally는 생략 가능

3
```
try{
  System.out.println(3/0);
  System.out.println("프로그램 종료");
}
catch(ArithmeticException e){
  System.out.println
          ("숫자는 0으로 나눌 수 없습니다.");
  System.out.println("프로그램 종료");
}
```

정상 수행

예외 발생

4
```
try{
  System.out.println(3/0);
}
catch(ArithmeticException e){
  System.out.println
          ("숫자는 0으로 나눌 수 없습니다.");
}
finally{
  System.out.println("프로그램 종료");
}
```

정상 + 예외

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

☞ 예외 처리 프로세스

```
try{
  System.out.println(3/0);
}


catch(ArithmethicException e){
  System.out.println
            ("숫자는 0으로 나눌 수 없습니다.
                                    ");
}


finally{
  System.out.println("프로그램 종료");
}
```

1

try block에서 예외발생

2

JVM

발생한 예외클래스 객체 생성
(ArithmeticException excep
 = new ArithmeticException())

3

생성객체를 받을 수 있는 catch블록으로 전달

4

개념상 JVM은 해당 예외 객체를 매개변수로 하는 catch(.) 메서드를 호출하는 개념
(실제 메서드는 절대 아님)

5

TIP

- 만일 JVM에서 생성된 예외객체를 받을 수 있는 catch(...){ } 블록이 없는 경우 예외처리가 되지 않음

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

☞ 다중 예외 처리 → 1

- try-catch 블록에서 catch블록은 여러 개 사용 가능
- 발생한 예외 객체를 받을 수 있는 catch 블록 실행

2

```
try{
  System.out.println(3/0);
}
catch(ArithmeticException e){
  System.out.println
          ("숫자는 0으로 나눌 수 없습니다.");
}
finally{
  System.out.println("프로그램 종료");
}
```

3

```
try{
  int a = Integer.parseInt("20A");
}
catch(NumberFormatException e){
  System.out.println
              ("숫자로 변환할 수 없습니다.");
}
finally{
  System.out.println("프로그램 종료");
}
```

4

```
try{
  System.out.println(3/0);
  int a = Integer.parseInt("20A");
}

catch(ArithmeticException e){
  System.out.println
          ("숫자는 0으로 나눌 수 없습니다.");
}

catch(NumberFormatException e){
  System.out.println
              ("숫자로 변환할 수 없습니다.");
}

finally{
  System.out.println
                  ("프로그램을 종료합니다.");
}
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

TIP

1 - catch 블록은 if-else if 구문처럼 위에서 부터 순차적으로 확인하여 하나의 catch블록 만을 실행

☞ 다중 예외 처리 → try-catch 블록에서 catch블록은 여러 개 사용 가능

```
try{
  int a = Integer.parseInt("20A");
  System.out.println(3/0);
}

catch(Exception e){
  System.out.println
          ("숫자는 0으로 나눌 수 없습니다.");
}

catch(NumberFormatException e){
  System.out.println
              ("숫자로 변환할 수 없습니다.");
}
```

2

3 JVM NumberFormatException 객체생성

4 Exception은 모든 예외클래스의 부모 클래스이므로 모든 예외 객체를 받을 수 있음 (if문의 else{}의 개념)

이 블록은 절대 실행될 수 없음

5 **해결방법**

- catch 블록의 순서를 변경

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

☞ 다중 예외 처리 → 1 하나의 catch블록에서 여러 개의 예외처리 가능
예외 타입을 OR (|) 로 연결

2

```
try{
   예외 발생 가능 영역
}
catch(예외타입A | 예외타입B 참조변수명){
   예외 처리 영역
}
```

OR (|) 로 연결

3 처리 내용 동일

```
try{
   System.out.println(3/0);
   int a = Integer.parseInt("20A");
}

catch(ArithmeticException e){
   System.out.println
                    ("프로그램을 종료합니다.");
}

catch(NumberFormatException e){
   System.out.println
                    ("프로그램을 종료합니다.");
}
```

4 OR(|) 연산자를 이용하여 하나의 catch 블록에서 두개 이상의 예외처리 가능

```
try{
   System.out.println(3/0);
   int a = Integer.parseInt("20A");
}

catch(ArithmeticException | NumberFormatException e){
   System.out.println("프로그램을 종료합니다.");
}
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

1 ☞ 리소스 자동해제 예외 처리 (try with resource)
→ JDK 1.7 이후 리소스를 자동으로 해지할 수 있는 문법구조

2
```
try (리소스 자동해제가 필요한 객체 생성){
  //예외 발생 가능 코드
}
catch(예외클래스이름 참조변수명){
  //해당예외가 발생한 경우 처리블록
}
finally{
  //예외 발생여부에 상관없이 무조건 실행블록
}
```

3
```
InputStreamReader is = null;
try {
  is = new InputStreamReader(System.in);
  System.out.println(is.read());
} catch (IOException e) {
  //예외처리
} finally {
  if(is!=null) {
    try {
      is.close();
    } catch (IOException e) {
      //예외처리
    }
  }
}
```

5
예외발생 여부와 관계 없이 try-catch 완료 후 자동으로 is.close() 실행

리소스 자동해제 객체는 반드시 close 메서드를 포함하여야 함 = 리소스 자동해제 객체는 AutoCloseable 인터페이스 구현하여야 함

4 resource 관리 / network, db, io는 자원해제 필요

```
try (InputStreamReader is = new InputStreamReader(System.in);){
  System.out.println(is.read());
}

catch (IOException e) {
  //예외처리
}
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

```
public static void main(String[] args) {
  //참고. System.in은 리소스를 해지하면 이후에는 콘솔 입력 불가
  //#1. try with resource 구문을 이용해서 자동으로 해제
  try (InputStreamReader isr1=new InputStreamReader(System.in);){
    char input = (char)isr1.read();
    System.out.println("입력글자 = "+input);
  } catch (IOException e) {
    e.printStackTrace();
  }

  //#2. 리소스를 사용하고 finally에서 리소스 해제하기
  InputStreamReader isr2=null;
  try {
    isr2=new InputStreamReader(System.in);
    char input = (char)isr2.read();
    System.out.println("입력글자 = "+input);
  } catch (IOException e) {
    e.printStackTrace();
  } finally {
    if(isr2!=null) {
      try {
        isr2.close();
      } catch (IOException e) { }
    }
  }
}
```

1

2

자동해제

3

사용불가

4

```
a
입력글자 = a
java.io.IOException: Stream cl
        at java.io.BufferedInp
        at java.io.BufferedInp
        at sun.nio.cs.StreamDe
        at sun.nio.cs.StreamDe
        at sun.nio.cs.StreamDe
        at sun.nio.cs.StreamDe
        at sun.nio.cs.StreamDe
        at java.io.InputStrean
        at pack01_exceptionhar
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 및 예외 처리

☞ 리소스 자동해제 예외 처리 (try with resource)
→ JDK 1.7 이후 리소스를 자동으로 해지할 수 있는 문법구조

클래스 직접 정의

1
```
class A implements AutoCloseable{
  String resource ;
  A(String resource) {
    this.resource=resource;
  }
  void abc() throws Exception {
  }

  @Override
  public void close() throws Exception {
    if(resource !=null) {
      resource=null;
      System.out.println("리소스 해제");
    }
  }
}
```

try catch 블록 사용을 위해서 추가

2
```
//#1. 리소스를 사용하고 finally에서 리소스 해제하기
A a1 = null;
try {
  a1=new A("특정파일");
  a1.abc();
} catch (Exception e) {
  System.out.println("예외처리");
} finally {
  if(a1.resource!=null)
    try {
      a1.close();
    } catch (Exception e) {}
}
```

3
```
//#2. try with resource 구문을 이용해서 자동으로 해제
try (A a2 = new A("특정파일");){
  a2.abc();
} catch (Exception e) {
  System.out.println("예외처리");
}
```

Copyright 2020. Donghyung Kim, All rights reserved

The End

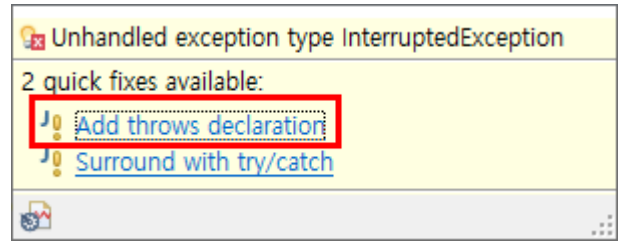
Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception)의 전가(throws)

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception)의 전가(throws)

2



☞ 예외의 전가(throws)

1 → 예외처리를 자신이 호출된 지점으로 전가 (이 경우 예외처리는 전가받은 상위위치에서 처리)
→ 메서드이름(...) **throws** 예외클래스

CASE #1

3 **bcd() 메서드가 스스로 예외를 처리한 경우**

5
```
void abc(…) {
  bcd(…);
}
```

1. 호출

4
```
void bcd(…) {
  try{
    예외가능블록;
  }
  catch(예외클래스타입 참조변수){
    예외처리
  }
}
```

2. 예외처리

CASE #2

6 **bcd()가 자신을 호출한 abc()에 예외를 전가한 경우**

이 경우 abc() 메서드가 예외를 처리하여야 함

8
```
void abc(…) {
  try{
    bcd();
  }
  catch(예외클래스타입 참조변수){
    예외처리
  }
}
```

3. 예외처리

1. 호출

7
```
void bcd(…) throws 예외클래스타입
{

  예외가능블록;

}
```

2. 예외전가

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception)의 전가(throws)

☞ 예외의 전가(throws)

1

```
//#1. 하위 메서드에서 직접 예외처리를 하는 경우
class A {

  void abc() {
    bcd();
  }

  void bcd() {
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      //예외처리 구문
    }
  }
}
```

2

```
//#2. 호출메서드로 예외를 전가한 경우
class B {

  void abc() {
    try {
      bcd();
    } catch (InterruptedException e) {
      //예외처리 구문
    }
  }

  void bcd() throws InterruptedException {
    Thread.sleep(1000);
  }
}
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception)의 전가(throws)

☞ 예외의 전가(throws)

1 → **Thread.sleep()메서드 호출시 예외처리(try-catch)가 필요했던 이유**

2

```
void abc(…) {
  try{
    Thread.sleep(1000);
  }catch (InterruptedException e) {
    //예외처리
  }
}
```

sleep() 메서드가 InterruptedException 예외를 전가
sleep 메서드를 호출하는 메서드는 예외를 처리 또는 전가하여야 함

3

sleep

```
public static void sleep(long millis)
                  throws InterruptedException
```

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers. The thread does not lose ownership of any monitors.

Parameters:

`millis` - the length of time to sleep in milliseconds

Throws:

`IllegalArgumentException` - if the value of millis is negative

`InterruptedException` - if any thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

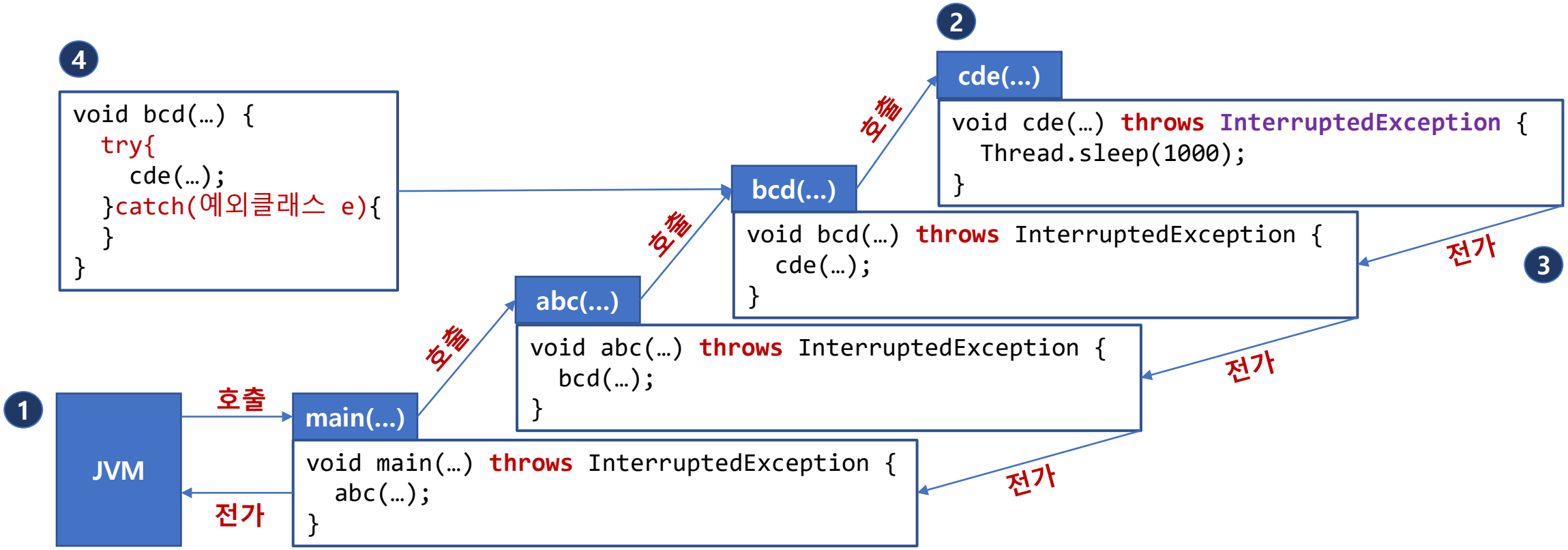
4 **Question**

- 상위 메서드들이 예외를 처리하지 않고 계속 전가만 한다면??

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception)의 전가(throws)

☞ 예외의 전가(throws)



Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception)의 전가(throws)

☞ 예외의 전가(throws)

1 **Question**

- 상위 메서드들이 예외를 처리하지 않고 계속 전가만 한다면??

2 **JVM 까지 전달**
예외의 원인 출력 + 프로그램 종료

3
```
public class EX10_ThrowsException_2 {
  public static void main(String[] args) throws Exception{
    Class cls = Class.forName("java.lang.Object2");
  }
}
```

4
```
Exception in thread "main" java.lang.ClassNotFoundException: java.lang.Object2
        at java.net.URLClassLoader.findClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Unknown Source)
        at pack01_exceptionhandling.sec03_ThrowsException.EX10_ThrowsException_
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception)의 전가(throws)

☞ 예외의 다중전가(throws) → 여러 개의 예외전가 가능

2

```
void abc(){
  try{
    bcd();
  }
  catch(ClassNotFoundException e){
    //예외처리
  }
  catch(InterruptedException e){
    //예외처리
  }
}
```

호출 ↓ ↑ 예외전가

1

```
void bcd() throws ClassNotFoundException, InterruptedException {

  Class cls = Class.forName("java.lang.Object");
  Thread.sleep(1000);

}
```

Copyright 2020. Donghyung Kim, All rights reserved

The End

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 클래스 사용자 정의

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 클래스 사용자 정의

1 ☞ 사용자 정의 예외 클래스 작성 및 발생방법

2 **STEP#1** - **사용자 정의 예외 클래스 작성방법 (Exception/RuntimeException 상속 (생성자 2개 지정))**

3 **#1. Exception 상속 :**
일반예외(Checked Exception)으로 생성

4 **#2. RuntimeException 상속 :**
실행예외(UnChecked Exception)으로 생성

5

```
class MyException extends Exception {

  MyException(){
  }

  MyException(String s){
    super(s); //부모생성자호출
  }

}
```

예외 메세지

6

```
class MyRTException extends RuntimeException {

  MyRTException(){
  }

  MyRTException(String s){
    super(s); //부모생성자호출
  }

}
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 클래스 사용자 정의

☞ 사용자 정의 예외 클래스 작성 및 발생방법

1 **STEP#2** - **사용자 정의 예외 클래스 객체 생성**

```
MyException me1 = new MyException();
MyException me2 =
             new MyException("예외메세지");
```

```
MyRTException mre1 = new MyRTException();
MyRTException mre2 =
                 new MyRTException("예외메세지");
```

2 **STEP#3** - **예외 발생 시키기** — **throw 예외객체**

```
throw me1;
throw me2;

throw new MyException();
throw new MyException("예외메세지");
```

예외처리를 하지 않는 경우 오류 발생

```
throw mre1;
throw mre2;

throw new MyRTException();
throw new MyRTException("예외메세지");
```

예외처리를 하지 않아도 오류 발생하지 않음

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 클래스 사용자 정의

☞ 사용자 정의 예외 클래스 작성 및 발생방법

1 - 예외를 발생시킨 메서드는 **방법#1. 해당 예외를 처리하거나 방법#2. 상위 메서드로 전달하여야 함**

2 **방법#1** 예외를 직접 처리

```
void abc_1(int num) {
  try {
    if (num>=70)
      System.out.println("정상동작");
    else
      throw new MyException();
  }
  catch(MyException e) {
    System.out.println("예외처리");
  }
}

void bcd_1() {
  abc_1(65);
}
```

3 **방법#2** 예외를 상위로 전가
상위 메서드에서 예외 처리

```
void abc_2(int num) throws MyException {
  if (num>=70)
    System.out.println("정상동작");
  else
    throw new MyException();
}


void bcd_2() {
  try {
    abc_2(65);
  } catch (MyException e) {
    System.out.println("예외처리");
  }
}
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 클래스 사용자 정의

TIP

- 모든 예외 클래스는 Throwable 클래스를 상속하며 getMessage(), printStackTrace()는 Throwable 클래스의 메서드임

1

☞ 예외 클래스의 대표적인 메서드

- **예외 메시지** 2 `String getMessage()` : 예외 발생시 **생성자로 넘긴 메시지**를 리턴

3

```
//#1. 예외객체 생성시 메시지를 전달하지 않은 경우

try {
  throw new Exception();
} catch(Exception e) {
  System.out.println(e.getMessage()); //null
}

//#2. 예외객체 생성시 메시지를 전달한 경우

try {
  throw new Exception("예외 메시지");
} catch(Exception e) {
  System.out.println(e.getMessage()); //예외 메시지
}
```

4

```
null
예외 메시지
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 클래스 사용자 정의

TIP

- 모든 예외 클래스는 Throwable 클래스를 상속하며 getMessage(), printStackTrace()는 Throwable 클래스의 메서드임

☞ 예외 클래스의 대표적인 메서드

1

- **예외 발생 경로 출력** `void printStackTrace()` : **예외 발생이 이루어지는 경로** (호출순서)를 출력

3

```
class A{
  void abc() throws NumberFormatException {
    bcd();
  }
  void bcd() throws NumberFormatException {
    cde();
  }
  void cde() throws NumberFormatException {
    int num = Integer.parseInt("10A");
  }
}
```

3

```
public static void main(String[] args) {
  //#1.객체 생성
  A a = new A();
  //#2. 메서드 호출 + 예외처리
  try {
    a.abc();
  } catch (NumberFormatException e) {
    e.printStackTrace();
  }
}
```

Message

4

```
java.lang.NumberFormatException: For input string: "10A"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:580)
        at java.lang.Integer.parseInt(Integer.java:615)
        at pack01_exceptionhandling.sec04_UserException.EX14_ExceptionMethods_2.A.cde(EX14_ExceptionMethods_2
        at pack01_exceptionhandling.sec04_UserException.EX14_ExceptionMethods_2.A.bcd(EX14_ExceptionMethods_2
        at pack01_exceptionhandling.sec04_UserException.EX14_ExceptionMethods_2.A.abc(EX14_ExceptionMethods_2
        at pack01_exceptionhandling.sec04_UserException.EX14_ExceptionMethods_2.EX14_ExceptionMethods_2.main(
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 클래스 사용자 정의

☞ 사용자 정의 예외 클래스 작성 및 발생방법 사용 예

1 **점수는 0~100까지만 유효 / 이외의 점수는 예외 발생**

점수가 음수인 경우 발생하는 예외 클래스 생성

2

```
class MinusException extends Exception {

  MinusException(){
  }
  MinusException(String s){
    super(s); //부모생성자호출
  }

}
```

점수가 100을 초과하는 경우 발생하는 예외 클래스 생성

3

```
class OverException extends Exception {

  OverException(){
  }
  OverException(String s){
    super(s); //부모생성자호출
  }

}
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 클래스 사용자 정의

☞ 사용자 정의 예외 클래스 작성 및 발생방법 사용 예

점수는 0~100까지만 유효 / 이외의 점수는 예외 발생

1 **점수가 0~100 범위가 아닌 경우 예외를 발생하는 메서드**

2

```
class A {
  void checkScore(int score) throws MinusException, OverException {

    if(score<0) {
      throw new MinusException("예외: 음수값 입력");
    }
    else if(score>100){
      throw new OverException("예외: 100점 초과");
    }
    else {
      System.out.println("정상적인 값입니다");
    }

  }
}
```

Copyright 2020. Donghyung Kim, All rights reserved

예외(Exception) 클래스 사용자 정의

☞ 사용자 정의 예외 클래스 작성 및 발생방법 사용 예

점수는 0~100까지만 유효 / 이외의 점수는 예외 발생

1 **A 객체의 checkScore 메서드를 호출하여 예외처리하는 메서드**

2

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

  A a = new A();

  try {
    a.checkScore(85);  //정상적인 값입니다
    a.checkScore(150); //예외발생
  }
  catch (MinusException | OverException e) {
    System.out.println(e.getMessage());
  }

}
```

```
정상적인 값입니다
예외:100점 초과
```

Copyright 2020. Donghyung Kim, All rights reserved

The End

Copyright 2020. Donghyung Kim, All rights reserved