INTERFACES
EXCEPTION HANDLING

18TH LECTURE

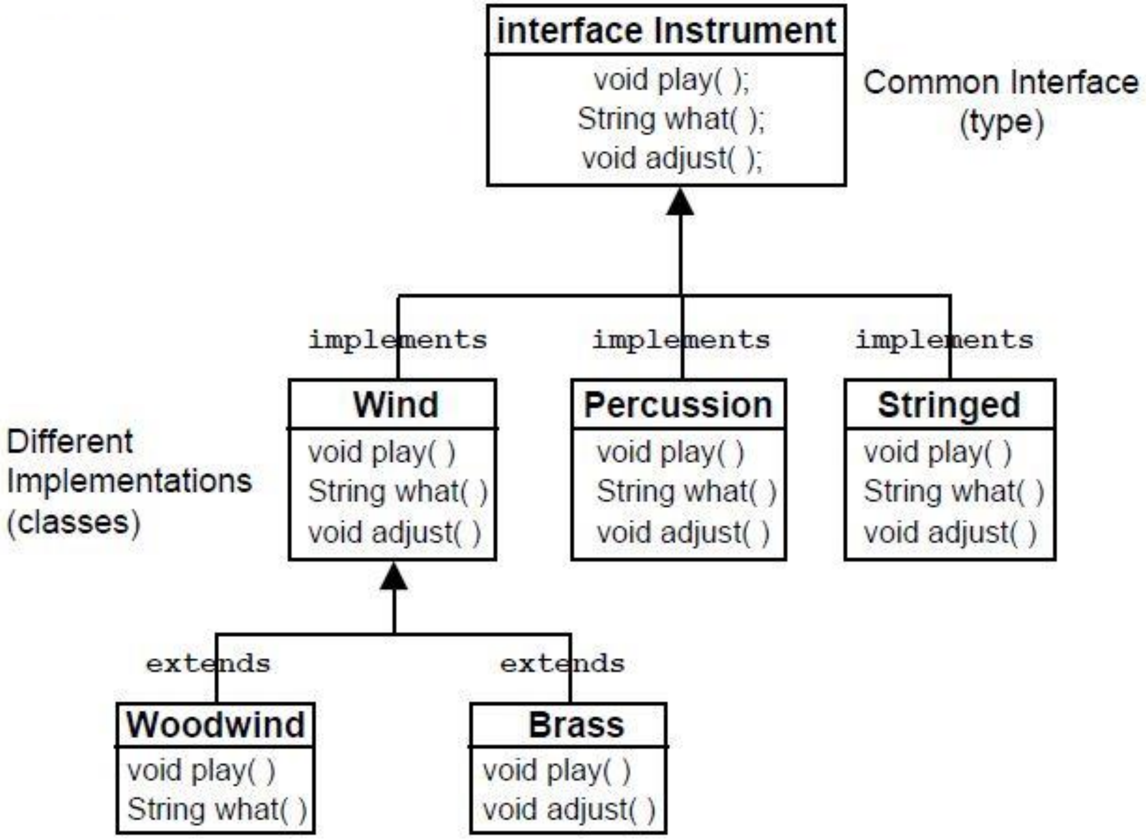
엄현상(Eom, Hyeonsang)
School of Computer Science and Engineering
Seoul National University

Outline

- **Interfaces**
  - An Instrument interface
  - “Multiple Inheritance” in Java
  - Java “Multiple Inheritance”
- **Error Handling with Exceptions**
  - The problem
  - What’s an exception?
  - Basic Exception / Catching an Exception
  - The Exception Specification
  - Creating your own exceptions
  - Catching any Exception
  - Rethrowing an Exception
  - RuntimeException
  - One more factor: finally
  - What’s “finally” For?
  - Exceptions in Constructors
  - Exception Matching
  - Catching Base-Class Constructor Exceptions
  - “Inheritance” of Exceptions
  - Overhead
  - Guidelines
  - Summary

Interfaces

- Can't have any fields or method definitions



An Instrument interface

- No “concrete” elements in interface
- You don’t extend, you implement

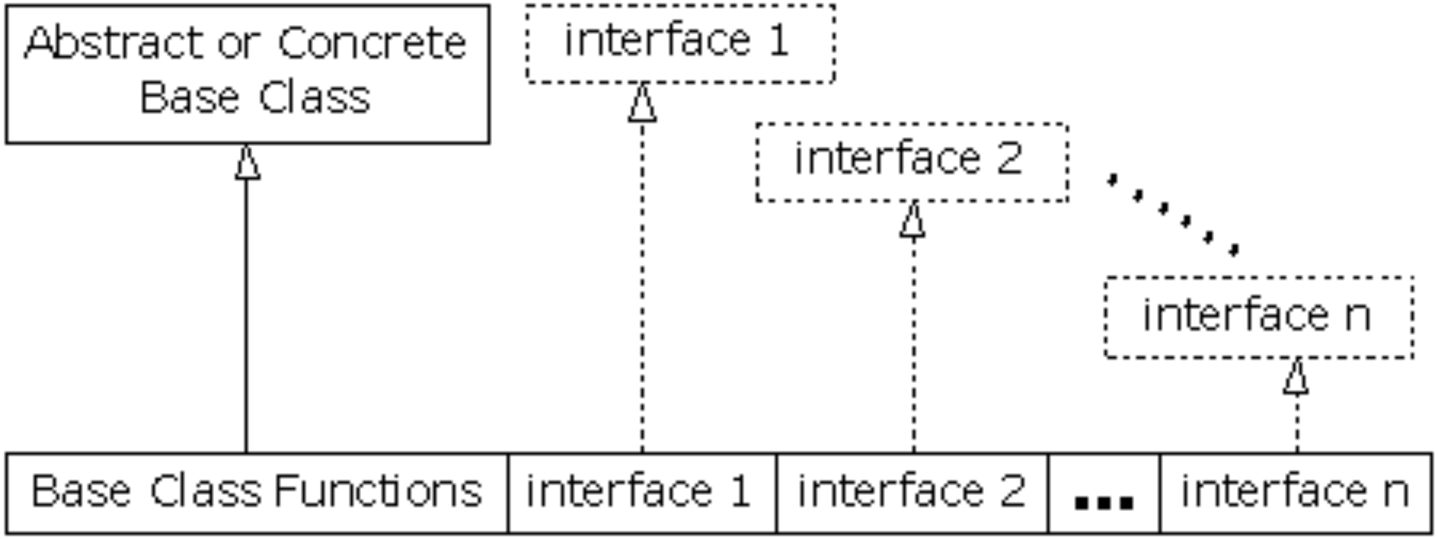
interface는 자동적으로 public이 되는거다.

```
import java.util.*;

interface Instrument {
  // Compile-time constant:
  int i = 5; // static & final
  // Cannot have method definitions:
  void play(); // Automatically public
  String what();
  void adjust();
}

class Wind implements Instrument {
  public void play() {
    System.out.println("Wind.play()");
  }
  public String what() { return "Wind"; }
  public void adjust() {}
}
```

“Multiple Inheritance” in Java

- New class has combined interfaces of all types
  - But using only one physical implementation: that of the concrete base class

여러 interfacee implement 가능



Java "Multiple Inheritance"

- To add extra interfaces
  - *Not* to combine implementations (using composition for that)
- Using it if you need to upcast to more than one base type
- Guideline
  - Using interfaces when possible, avoiding abstract classes
  - You never know when you'll need to combine interfaces; any sort of concreteness prevents it

Error Handling with Exceptions

- Java
  - "Badly-formed code will not be run"
- Not all errors can be caught at compile time
- Run-time error handling integrated into the core of the language, enforced by the compiler
- Can't get too far learning the language without it

The problem

- Coping with errors during program execution
- Errors can be caused by
  - Program logic
    - I.e., exceeding array bounds
    - Can be prevented by the programmer
  - Status of the environment
    - I.e., network goes down
      - Cannot be prevented by the programmer

What’s an exception?

- Exception
  - A type of object that signals an error condition and provides information about the error
- Once an exception is generated, control is passed *up the call stack* to a specific handler
  - You can have as many handlers as you want, for different exceptions and/or at different levels
- Java exceptions cannot be ignored

Basic Exception

- *Exceptional Condition*
  - not enough info in the current context to continue processing
- **throw** an exception:

```
if(t == null)
throw new NullPointerException();
```

- Exception arguments

```
if(t == null)
throw new NullPointerException("t=null");
```

  - Like any other constructor
  - Info can be extracted later

Catching an Exception

- **try** block
  - A guarded region

```
try {
// Code that may generate exceptions
} catch(Type1 id1) {
// Handle exceptions of Type1
} catch(Type2 id2) {
// Handle exceptions of Type2
} catch(Type3 id3) {
// Handle exceptions of Type3
}
// etc...
```

The Exception Specification

런타임 은 포함되어 있다?
모든 런타임 예외 throw 가능하고
명시되지 않아도,, 그렇게 간주해랏

java.io.IOException 통해서 가능

void f() throws TooBigException { //...

java Object
java lang Throwable
java.lang.Exception
java.lang.RuntimeException

- If you say **void f() {}**
- It means that no exceptions (*except* for those derived from the special class **RuntimeException**) may be thrown
- Compiler verifies exception specifications!
- This guarantees that all (checked) exceptions will get caught somewhere

Creating your own exceptions

```
class MyException extends Exception {
  public MyException() {}
  public MyException(String msg) {
    super(msg);
  }
}
```

이미 build in package에 들어가 있다.

```
>>
Throwing MyException from f()
MyException
        at
FullConstructors.f(FullConstructors.java:16)
        at
FullConstructors.main(FullConstructors.java:24)
Throwing MyException from g()
MyException: Originated in g()
        at
FullConstructors.g(FullConstructors.java:20)
        at
FullConstructors.main(FullConstructors.java:29)
```

```
public class FullConstructors {
  public static void f() throws MyException {
    System.out.println(
      "Throwing MyException from f()");
    throw new MyException();
  }
  public static void g() throws MyException {
    System.out.println(
      "Throwing MyException from g()");
    throw new MyException("Originated in g()");
  }
  public static void main(String[] args) {
    try {
      f();
    } catch(MyException e) {
      e.printStackTrace(System.err);
    }
    try {
      g();
    } catch(MyException e) {
      e.printStackTrace(System.err);
    }
  }
} ///:~
```

2

System…

java. lang.System
(Object class가 extend 한다.)
자동으로 포함, import 되어 있으므로, 포함시킬 필요 x
그냥 바로 System.out 등등 사용 가능하다.

```
Class System
     public static final InputStream in;
                              PrintStream out;
                              (same ) err;
```

(System class 내에 object 로 inputStream
OutputStream 정의되어 있다.)

****

try block 내의 funcion call 시
exception handling 시 동일한
exception handling 가능하다

call chain 에서 상위 try 그리고
catch는 더 넓은 범위에서 하겠쥬

‘

Catching any Exception

- All the exceptions you need to worry about
- Being derived from **Exception**

```
catch(Exception e) {
System.out.println("Caught exception");
}
```

- Special system errors are derived from**Error**
- Program bugs: **RuntimeException**
  - –These are thrown automatically for run-time programming errors

Rethrowing an Exception

```
catch(Exception e) {
    System.out.println("Exception was thrown");
    throw e;
}
```

- Performing anything you can locally, then letting a global handler perform more appropriate activities

What’s in a name?

- Name of the exception is typically the most important thing about it
- Names tend to be long and descriptive
- Code for the exception class itself is usually minimal
- Once you catch the exception you are usually done with it

RuntimeException

- Name is confusing, since every exception is thrown at runtime
- Base class for all errors generated by programming mistakes that *appear* at runtime
  - **NullPointerException**, **runrime exception의 sub이다.**
  - **ArrayIndexOutOfBoundsException**,
  - **IllegalArgumentException**, etc.

indexOut OfBoundsClass의 sub이다.
즉, 실제로는 이 class를 extend하는 것...!
array가 아니라~~~

- Do not need to include **RuntimeException** classes in the exception specification

One more factor: finally

- At least one catch or finally clause must be present

```
try {
  // The guarded region: Dangerous activities
  // that might throw A, B, or C
} catch(A a1) {
  // Handler for situation A
} catch(B b1) {
  // Handler for situation B
} catch(C c1) {
  // Handler for situation C
} finally {
  // Activities that happen every time
}
```

Try block (mandatory)

Catch clauses

Finally clause

C++에는 없음.무조건 실행해야 한다.
즉, tray catch 하고 항상 finally는 실행된다.
적어도 하나의 catch나 finally 있어야 한다.
(try 내에)

What’s “finally” For?

- Always getting called, regardless of what happens with the exception and where it’s caught
- To set something *other* than memory back to its original state (GC handles memory) (close files, network connections, etc.)

```
class Switch {
        boolean state = false;
        boolean read() { return state; }
        void on() { state = true; }
        void off() { state = false; }
}
```

```
public class WithFinally {
  static Switch sw = new Switch();
  public static void main(String[] args) {
    try {
      sw.on();
      // Code that can throw exceptions...
      OnOffSwitch.f();
    } catch(OnOffException1 e) {
      System.err.println("OnOffException1");
    } catch(OnOffException2 e) {
      System.err.println("OnOffException2");
    } finally {
      sw.off();
    }
  }
} ///:~
```

false 로 switch object original 로 만들어주는 것,,,, 마지막에는 꼭 이렇게 set 마무리 ㅎㅅㅎ

```
class FourException extends Exception {}

public class AlwaysFinally {
  public static void main(String[] args) {
    System.out.println(
      "Entering first try block");
    try {
      System.out.println(
        "Entering second try block");
      try {
        throw new FourException();
      } finally {
        System.out.println(
          "finally in 2nd try block");
      }
    } catch(FourException e) {
      System.err.println(
        "Caught FourException in 1st try block");
    } finally {
      System.err.println(
        "finally in 1st try block");
    }
  }
} ///:~
```

```
>>
Entering first try block
Entering second try block
finally in 2nd try block
Caught FourException in 1st try block
finally in 1st try block
```

Exceptions in Constructors

```
import java.io.*;

class InputFile {
  private BufferedReader in;
  InputFile(String fname) throws Exception {
    try {
      in =
        new BufferedReader(
          new FileReader(fname));
      // Other code that might throw exceptions
    } catch(FileNotFoundException e) {
      System.err.println(
        "Could not open " + fname);
      // Wasn't open, so don't close it
      throw e;
    }
```

```
catch(Exception e) {
      // All other exceptions must
close it
      try {
        in.close();
      } catch(IOException e2) {
        System.err.println(
          "in.close() unsuccessful");
      }
      throw e; // Rethrow
    } finally {
      // Don't close it here!!!
    }
  }
```

타입에 따른 실행여부

Exception Matching

- Base-class handler will catch
- Derived-class object

```
class Annoyance extends Exception {}
class Sneeze extends Annoyance {}

public class Human {
  public static void main(String[] args) {
    try {
      throw new Sneeze();
    } catch(Sneeze s) {
      System.err.println("Caught Sneeze");
    } catch(Annoyance a) {
      System.err.println("Caught Annoyance");
    }
  }
} ///:~
```

catch가 실행이 되면,즉, sneeze가 실행이 되면,sneeze가 extend하는 annoyance 도 실행이 되는지?????

match가 되면 해당 catch clouse 실행이 되고, catch이후부터 실행이 된다. 하지만, sneeze가 없으면, annoyance도 실행 되겠쥬?
결론능 실행된다.!!!~(?)
+ NOPE
Error 발생 이미 Annoyance에 의해서 handled 된다.

Catching Base-Class Constructor Exceptions

- Cannot have *anything* before base-class constructor call, not even a **try** block
- Thus cannot catch base-class constructor exceptions in the derived-class constructor
- Must show exception in derived-class constructor exception specification

base class 에서 예외 발생하면,
derived 에 specification 있어야 하고,
그거 handling 하면 된다....

밑의 코드 부연,,,

Code Example

```
class Base {
    Base() throws CloneNotSupportedException {
        throw new CloneNotSupportedException();
    }
}
class Derived extends Base {
    Derived() throws CloneNotSupportedException, RuntimeException {}
    public static void main(String[] args) {
        try {
            Derived d = new Derived();
        }
        catch(CloneNotSupportedException e) {
            e.printStackTrace();
        }
        catch(RuntimeException re) {}
    }
}
```

runtiome exception 아니기 때문에,
explicit specification에 명시 해주어야 한다.

사실 쓸 필요 no
rebundant 이미
자동으로,,,

이건 가능..

http://stackoverflow.com

Code Example (Cont’d)

```
class Derived extends Base {
    Derived() throws CloneNotSupportedException {
        try {
            super();
        } catch (CloneNotSupportedException e) {
            System.out.println("We have indeed caught an exception from the "+
               "base-class constructor! The book was wrong!");
        }
    }
    public static void main(String[] args) {
        try {
            Derived d = new Derived();
        }
        catch(CloneNotSupportedException e) {
            e.printStackTrace();
        }
    }
}
```

base class 내의 exception을 derived 안에서 하려고 함 -> compile time error

super() 보다 try가 먼저 나오는 것 안됨.
이러한 시도 자체 불가능. 무조건 super() 먼저!

base -> derived 원칙 위배하면 안된다

default constructor가 없으면,
derived에서 해당 가장 먼저 argu 부른다.
super는 가장 처음 statement 여야함.

http://stackoverflow.com

“Inheritance” of Exceptions

- Base-class method throws an exception
  - Derived-class method may throw that exception or one derived from it
- Derived-class method
  - Throwing an exception that isn’t a type/subtype of an exception thrown by the base-class method

Overhead

- Exceptions are free as long as they don’t get thrown
- If they are thrown, very expensive
- Not using exceptions for normal flow of control
- Only using exceptions to indicate abnormal conditions

Guidelines

- Handling an exception
  - Only if you have enough information in the current context to correct the error (partially or totally)
  - Otherwise, just letting the exception propagate up
- Separating error handling code (which almost never runs) from code that represents the normal path of execution
  - Making code more readable

Guidelines Cont’d

- Handling tasks, not statements
  - Not encompassing every single statement in a try block
  - Instead, putting tasks inside of a try block, then handling each exception that can occur
- Using loops to retry
  - Like C++, no *resumption* in Java
  - If you need to retry, putting the exception handling inside a **do**…**while** loop

Guidelines Cont’d

- Using exceptions in constructors
  - People assume construction succeeds
- If you catch an exception, doing something with it
  - Not “stubbing it out” by having an empty
- Handler
  - This discards the exception; not robust coding
- Cleaning up using **finally**

Summary

- You have no choice in Java
    - You *must* catch exceptions
    - You *must* use exception specifications
    - The compiler enforces exception use
- A clean, straightforward error-handling model
    - You don't have to decide how to handle errors
    - You don't have to figure out how someone else handles errors
    - You don't worry about whether errors get handled
- Seemingly more work at first
    - Only because you've been ignoring errors!

array => as list?? 이걸로 하면, resize는 안된다
upcast 사용은 가능하지만, 기본적으로 fix 된 size라서
다시 resize는 안된답

container
set -> 중복 제거

collection -> List, Queue, Map 3가지를 말함

map 의 경우에는 key value pair에 대한 efficient 저장 등

Type safe container
runtime error 발생한다 왜? 놓침…
orange

arrau list object 만들 때, type을 명시해주어야 한다.
compile 시간에 apple class object 인데, 왜 orange add 하는지
오류를 나타내주겠쥬—

for(Appla c : apples)

size는 0 후에는 1 2 등등으로 object 만들어지겠쥬
각각 라인에 0 1. 2 나오겠쥬

class 명 object 이름, array list 명
등으로 for문 가능 ㅎㅎ
tostring invoke 되어서 출력된답

Grannysmith 의 경우는 출력값 어떻게 되나유 ㅎㅅㅎ
주소의 해쉬값(해쉬코드 method 의 return 값) + class 이름
도 나온답 !!!!!!

lisst A = linkedlist
list B = array

upcasting 시 주의할 점,
sarray linked 등에 있는 것 list에 없을 수 있다.
혼동의 여지 존재

Arrays 라는 object 만들어서 넣는 경우
///
list<Integet> list = Arrays.addlist()
의 경우 list.add 새로 size 늘리는 것 불가능하다
fix size이다. upcast기 때문에,,,???? (꼭 다시)

....

collection.add("rat")
collection.add("cat")
collection.add("dog")
collection.add("dog")
오름차순…? 으로 중복 없에>>
map 의 경우에는 association ..? 어떻게?
map.put("rat", "Fuzzy")
map.put("cat", "Rags")
.
.
이렇게 key value pair가 되어서 들어가게 되는 것
linked hash set은 들어온 순서로 중복 없앤다
treemap은 오름차순으로 키가 정렬되고, map의 경우는
같은 key에 대해서 두번 콜되면, 이전 value vs 이후 value
상식적으로 뒤의 값이 남아있다.

arraylist의 object를 만들어서 그 결과를 프린트 하는 것이다.

<arraylist>
[rat cat dog dog]
<linkedlist>
[dog, cat, rat,]
결과값
<ascending order>
[cat, dog, rat]
<linkedhash>
[rat, cat, dog]
<hashmap -hashset과 same>
[dogspot, CatRat, RatFuzzy,]
< ??? >
[catRat, dogspot, RatFuzzy]
<linkedhashmap>
[ratFuzzy, catRat, dogSpot]

각각 list 들 출력시 들어간 순서로 나오겠쥬
(list) Arraylist, linkedlist,
하지만, hashset treeset linkedhashset 등등의
set에 대해서는 약간 다르다. 복잡스
(set)으로 묶인거다. 다른 implementation 사용(object에 따라서)
collection object를 parameter로 갖는 list

String 비교시 아스키 코드값을 비교한다.
커진 값이 뒤에 오는 등으로 sorting하는 것이다

lineked hash는 들어온 순서로 보여준다.
tree는 ascending order
list same
map key value에 의해?
array 랑 linked는 결과값은 같으나,
array 기반인지, linked 기반인지에 따라 결과값 다르다

hashmap의 경우는 hash set과 same

key value association 된거는 달라지지 않는다.
나열되는 순서가 달라지는 것,

key는 중복되면 안되며, 중복되면 뒤에 들어오는 값이 쓰인다.
distink 해야한다.

텍스트