# 11. 예외 처리

학습목표

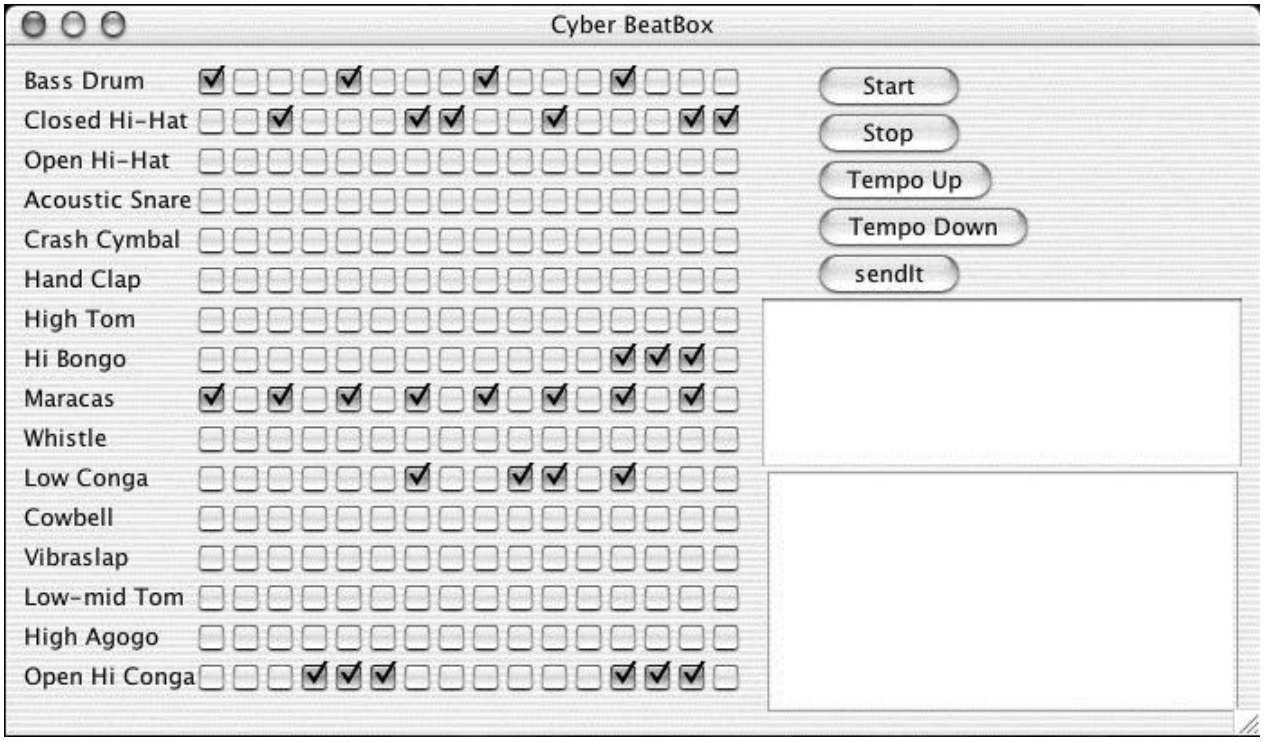
**음악 재생 프로그램**
**예외 처리 방법**
**try/catch 블록**
**예외 선언 방법**

# 위험한 행동

- 예상치 못한 상황
  - 파일이 없는 경우
  - 서버가 다운되는 경우
  - 장치를 사용할 수 없는 경우

- 이런 예외적인 상황을 처리하기 위한 방법이 필요합니다.
  - 자바의 예외 처리 메커니즘
  - try/catch 블록
  - 예외 선언

# 음악 재생 프로그램


Cyber BeatBox
Bass Drum
Closed Hi-Hat
Open Hi-Hat
Acoustic Snare
Crash Cymbal
Hand Clap
High Tom
Hi Bongo
Maracas
Whistle
Low Conga
Cowbell
Vibraslap
Low-mid Tom
High Agogo
Open Hi Conga
Start
Stop
Tempo Up
Tempo Down
sendIt

# JavaSound API

- JavaSound API
  - MIDI
    - 악기 디지털 인터페이스(Musical Instruments Digital Interface)
  - Sampled

# Sequencer

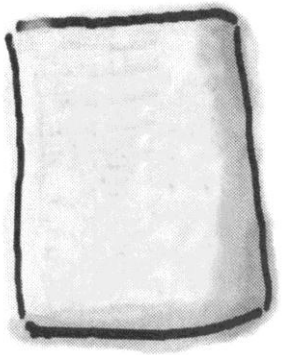
```
import javax.sound.midi.*;

public class MusicTest1 {
   public void play() {
      Sequencer sequencer = MidiSystem.getSequencer();
      System.out.println("We got a sequencer");
   }
   public static void main(String[] args) {
      MusicTest1 mt = new MusicTest1();
      mt.play();
   }
}
```

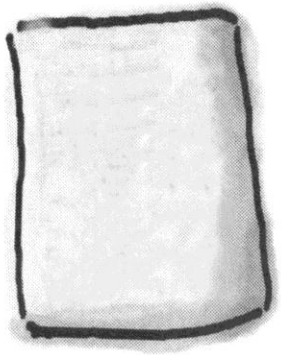
```
% javac MusicTest1.java
MusicTest1.java:5: unreported exception
javax.sound.midi.MidiUnavailableException; must be caught or declared
to be thrown
      Sequencer sequencer = MidiSystem.getSequencer();
                                                     ^
1 error
```

# 위험 요소가 있는 메소드

프로그래머

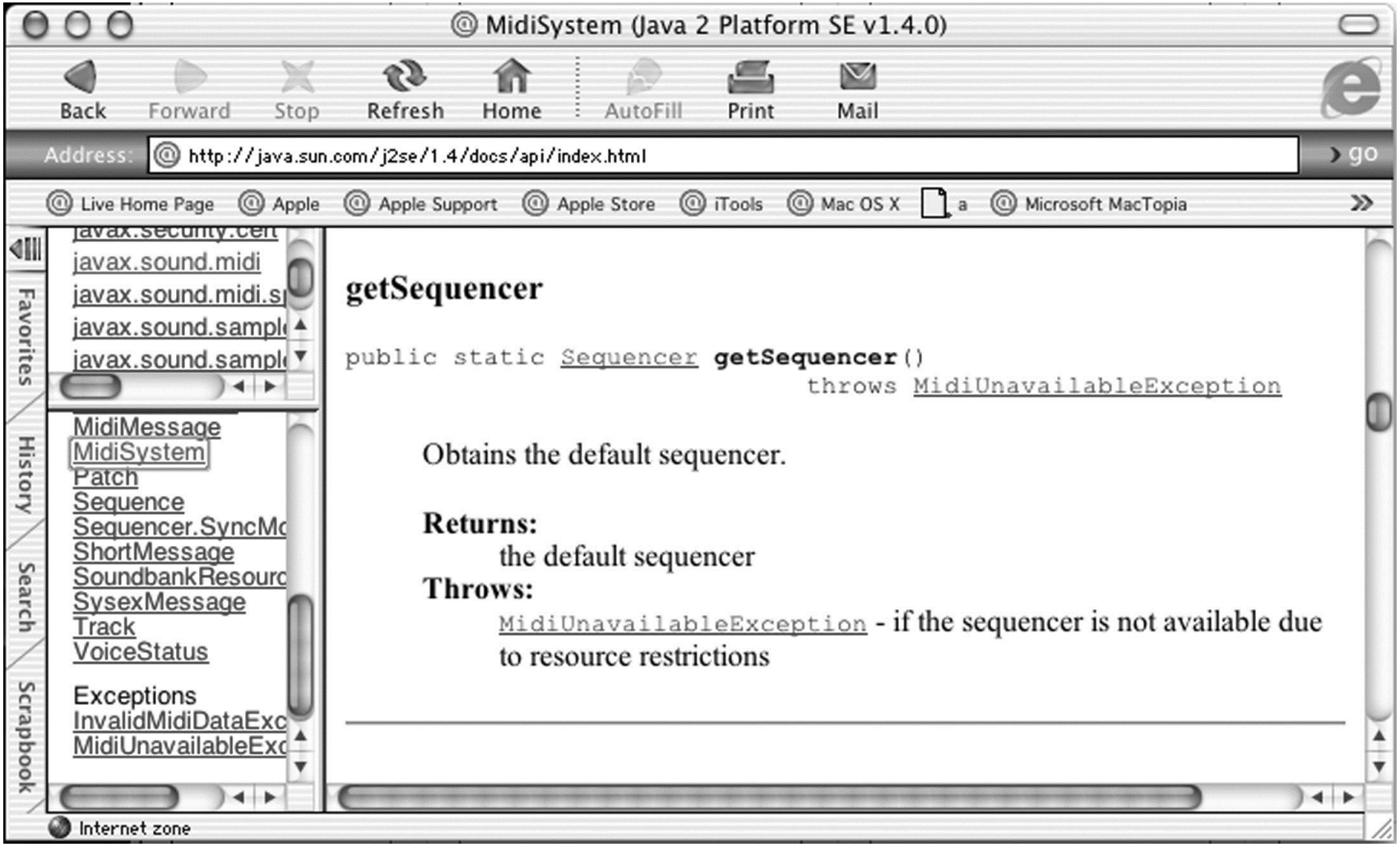


프로그래머가 만든 코드



코드에서 사용하는 메소드가 들어있는 클래스

# 예외 처리 메커니즘


MidiSystem (Java 2 Platform SE v1.4.0)
Back
Forward
Stop
Refresh
Home
AutoFill
Print
Mail
Address:
http://java.sun.com/j2se/1.4/docs/api/index.html
go
Live Home Page
Apple
Apple Support
Apple Store
iTools
Mac OS X
a
Microsoft MacTopia
javax.sound.midi
javax.sound.midi.sy
javax.sound.sampl
javax.sound.sampl
MidiMessage
MidiSystem
Patch
Sequence
Sequencer.SyncMo
ShortMessage
SoundbankResourc
SysexMessage
Track
VoiceStatus
Exceptions
InvalidMidiDataExc
MidiUnavailableExc
Favorites
History
Search
Scrapbook
getSequencer
public static Sequencer getSequencer()
throws MidiUnavailableException
Obtains the default sequencer.
Returns:
the default sequencer
Throws:
MidiUnavailableException - if the sequencer is not available due to resource restrictions
Internet zone

# try/catch 블록

- 예외를 처리할 것임을 알려주기 위한 용도로 쓰임

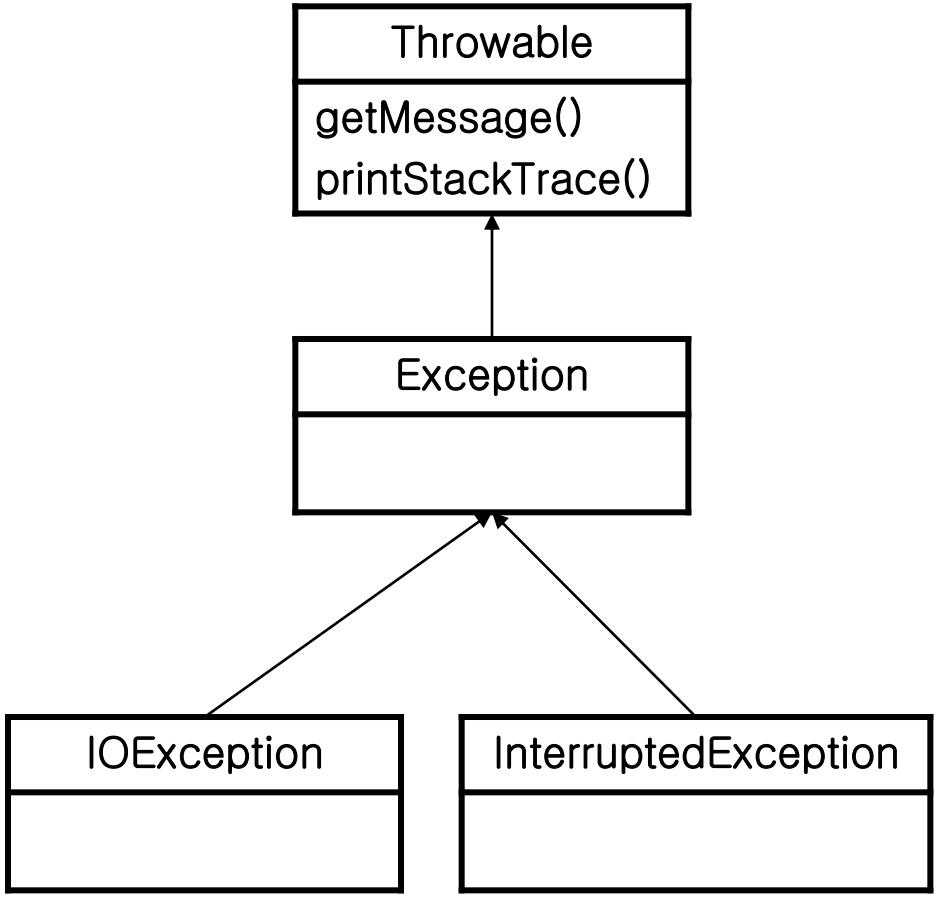
```
import javax.sound.midi.*;

public class MusicTest1 {
   public void play() {
      try {
         Sequencer sequencer = Midisystem.getSequencer();
         System.out.println("Successfully got a sequencer");
      } catch (MidiUnavailableException ex) {
         System.out.println("Bummer");
      }
}

   public static void main(String[] args) {
      MusicTest1 mt = new MusicTest1();
      mt.play();
   }
}
```

# Exception 클래스

```
try {
    // 위험한 일 처리
} catch (Exception ex) {
    // 문제 해결
}
```



# 예외 던지기와 잡기

- 예외를 던지는 코드

```
public void takeRisk() throws BadException {
    if (abandonAllHope) {
        throw new BadException();
    }
}
```

- 위험한 메소드를 호출하는 코드

```
public void crossFingers() {
    try {
        anObject.takeRisk();
    } catch (BadException ex) {
        System.out.println("Aaargh!");
        ex.printStackTrace();
    }
}
```

# 확인 예외

- 확인 예외(checked exception)
  - RuntimeException을 제외한 모든 예외
  - 코드에서 예외를 던진다면 반드시 메소드를 선언하는 부분에서 throws 키워드를 써야 함
  - 예외를 던지는 메소드를 호출하면 try/catch 블록으로 그 부분을 감싸거나 그 메소드에서도 예외를 선언해야 함

# 바보 같은 질문은 없습니다.

- NullPointerException, DivideByZero 같은 예외에 대해서는 왜 try/catch 블록을 사용하지 않나요?
  - RuntimeException 및 그 하위클래스에 속하는 예외는 컴파일러에서 잡아내지 않습니다.
  - 런타임 예외는 실행 중에 어떤 조건에 문제가 생기는 경우보다는 코드의 논리에 예측/예방할 수 없는 방식으로 문제가 생기는 경우에 발생합니다.
  - try/catch 블록은 예외적인 상황을 처리하기 위한 것이지 코드의 문제점을 보완하기 위한 것은 아닙니다.

# try/catch 블록의 흐름 제어

```
try {
    Foo f = x.doRiskyThing();
    int b = f.getNum();
} catch (Exception ex) {
    System.out.println("failed");
}
System.out.println("We made it!");
```

```
% java Tester
We made it!
```

```
% java Tester
failed
We made it!
```

# 무조건 실행할 내용

- 예외 발생 여부와 상관없이 무조건 실행할 코드는 finally 블록에...

```
try {
    turnOvenOn();
    x.bake();
} catch (BakingException ex) {
    ex.printStackTrace();
} finally {
    turnOvenOff();
}
```

```
try {
    turnOvenOn();
    x.bake();
    turnOvenOff();
} catch (BakingException ex) {
    ex.printStackTrace();
    turnOvenOff();
}
```

# 두 개 이상의 예외

- 예외를 여러 개 던진다면 모든 확인 예외를 잡아야 합니다.


```
public class Laundry {
   public void doLaundry() throws PantsException, LingerieException {
      // 두 가지 예외를 던질 수 있는 코드
   }
}

public class Foo {
   public void go() {
      Laundry laundry = new Laundry();
      try {
         laundry.doLaundry();
      } catch (PantsException pex) {   }
      } catch (LingerieException lex) {   }
   }
}
```


# 예외와 다형성

```
public void doLaundry throws ClothingException {
```


```
} catch (ClothingException cex) {
```

```
 } catch (ShirtException sex) {
```

# 작은 것부터 큰 것으로

```
try {
    laundry.doLaundry();
} catch(TeeShirtException tex) {


} catch(LingerieException lex) {


} catch(ClothingException cex) {


}
```


여러 개의 catch 블록을 쓸 때는
상/하위클래스 관계를 잘
따져봐야 합니다.

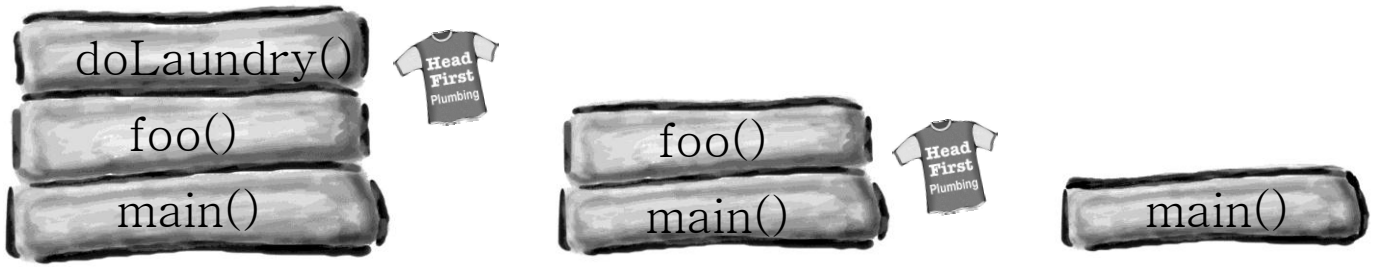
# 예외 선언

- try/catch 블록을 쓰는 대신 메소드에서 예외를 선언함으로써 예외 처리를 회피하는 방법도 있습니다.

```
public void foo() throws ReallyBadException {
    // try/catch 블록 없이 위험한 메소드 호출
    laundry.doLaundry();
}
```

# 예외 선언

```
public class Washer {
    Laundry laundry = new Laundry();
    public void foo() throws ClothingException {
        laundry.doLaundry();
    }
    public static void main(String[] args) throws ClothingException {
        Washer a = new Washer();
        a.foo();
    }
}
```



# 다시 음악 코드로...

```
import javax.sound.midi.*;

public class MusicTest1 {
   public void play() {
      try {
         Sequencer sequencer = MidiSystem.getSequencer();
         System.out.println("Successfully got a sequencer");
      } catch(MidiUnavailableException ex) {
         System.out.println("Bummer");
      }
   }
   public static void main(String[] args) {
      MusicTest1 mt = new MusicTest1();
      mt.play();
   }
}
```

# 예외와 관련된 규칙

1. try 없이 catch나 finally만 쓸 수는 없음

```
void go() {
    Foo f = new Foo();
    f.foof();
    catch(FooException ex) { }
}
```

2. try와 catch 사이에 코드를 집어넣을 수 없음

```
try {
    x.doStuff();
}
int y = 43;
} catch(Exception ex) { }
```

# 예외와 관련된 규칙

3. try 뒤에는 반드시 catch나 finally가 있어야 함

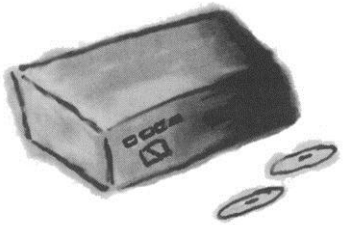
```
try {
    x.doStuff();
} finally { }
```

4. try 뒤에 finally만 있으면 예외 선언

```
void go() throws FooException {
    try {
        x.doStuff();
    } finally { }
}
```

# JavaSound API

- 음악을 재생하는 장치
  - Sequencer



- 재생할 음악
  - Sequence

- 실제 정보가 들어있는 부분
  - Track
- 실제 음악 정보
  - MidiEvent

# JavaSound API

1. 시퀀서 만들고 열기

```
Sequencer player = MidiSystem.getSequencer();
```

2. 새로운 시퀀스 만들기

```
Sequence seq = new Sequence(timing, 4);
```

3. 시퀀스에서 새로운 트랙 가져오기

```
Track t = seq.createTrack();
```

4. 트랙에 MidiEvent를 채우고 시퀀스를 시퀀서에 넘기기

```
t.add(myMidiEvent1);
player.setSequence(seq);
```

5. 시퀀서에 대해 play() 메소드 호출

```
player.start();
```

356 페이지에 있는 코드를 실행시켜봅시다.

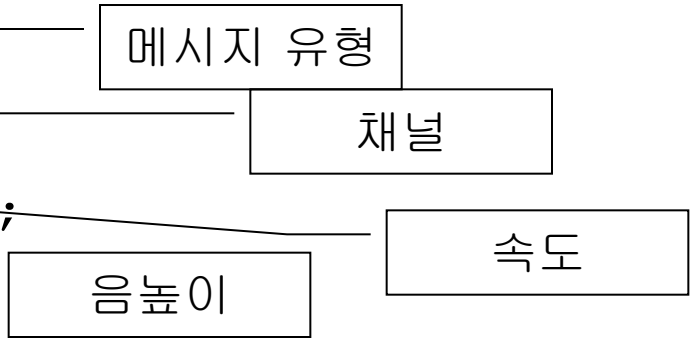
# MidiEvent

1. Message 만들기

```
ShortMessage a = new ShortMessage();
```

2. 메시지에 지시사항 넣기

```
a.setMessage(144, 1, 44, 100);
```




3. 새로운 MidiEvent 만들기

```
MidiEvent noteOn = new MidiEvent(a, 1);
```

4. MidiEvent를 트랙에 추가

```
track.add(noteOn);
```

# 숙제

- 본문을 꼼꼼하게 읽어봅시다.
- 연필을 깎으며 및 11장 끝에 있는 연습문제를 모두 각자의 힘으로 해결해봅시다.
- API 문서에서 이 장에 나와있는 클래스 및 메소드에 대한 내용을 직접 찾아봅시다.