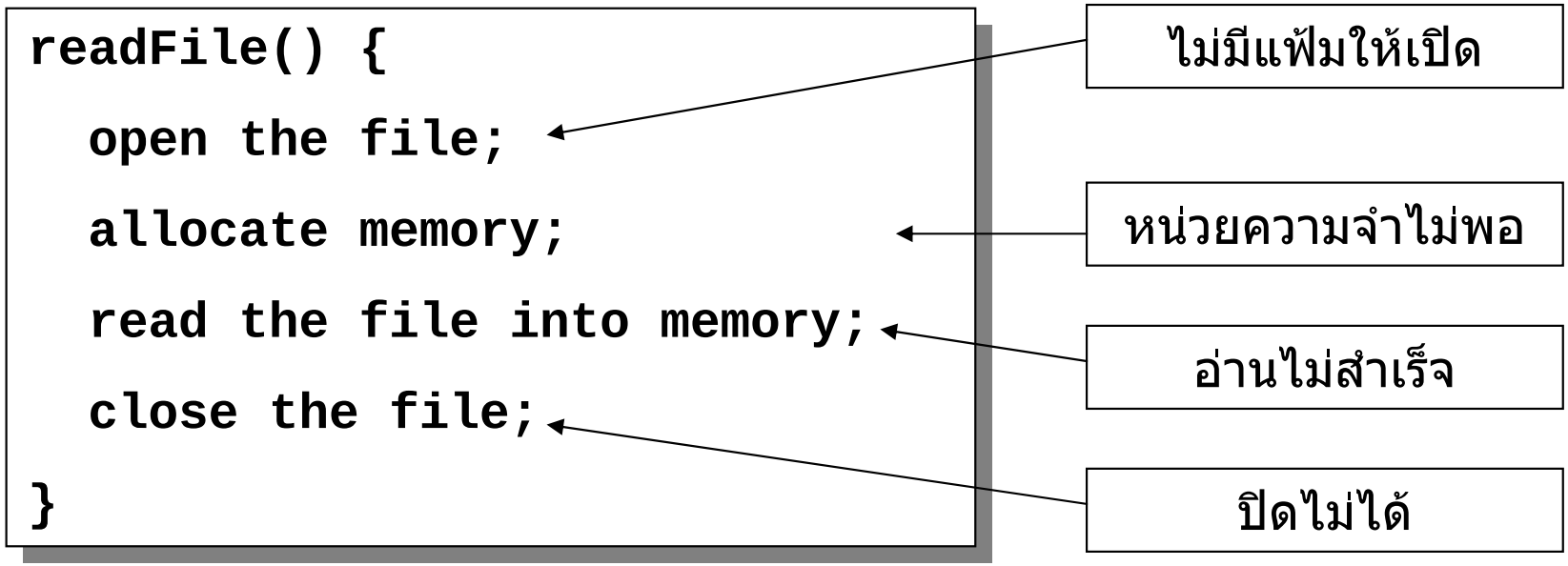
ภาษาจาวา – Exception & Assertion

สมชาย ประสิทธิ์จูตระกูล

อะไร ๆ ก็เกิดขึ้นได้



แบบเดิม ๆ

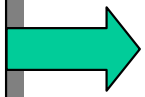
```
int readFile() {
  errCode = 0;
  c = open the file;
  if (c == theFileIsOpen) {
    c = allocate memory;
    if (c == gotEnoughMemory) {
      c = read the file into memory;
      if (c != readSuccess) errCode = -2;
    } else {
      errCode = -3;
    }
    c = close the file;
    if (c != theFileClose && errCode == 0) errCode = -4;
  } else {
    errCode = -1;
  }
  return errCode;
}
```

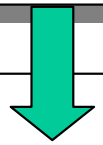
แบบใหม่ อ่านง่ายกว่า

```
readFile() {
  try {
    open the file;
    allocate that much memory;
    read the file into memory;
    close the file;
  } catch (fileOpenFailed) {
    doSomething;
  } catch (memoryAllocationFailed) {
    doSomething;
  } catch (readFailed) {
    doSomething;
  } catch (fileCloseFailed) {
    doSomething;
  }
}
```

Error Propagation

```
method1 {method2();}
method2 {method3();}
method3 {readFile();}
```





```
method1 {
  try {
    method2(); proceed();
  } catch (exception) {
    doErrorProcessing;
  }
}
method2 throws exception {
  method3(); proceed();
}
method3 throws exception {
  readFile(); proceed();
}
```

แบบใหม่

```
method1 {
  err = method2();
  if (err)
    doErrorProcessing;
  else
    proceed();
}
int method2 {
  err = method3();
  if (err)
    return err;
  else
    proceed();
}
int method3 {
  err = readFile();
  if (err)
    return error;
  else
    proceed();
}
```

แบบเดิม ๆ

Errors และ Exceptions

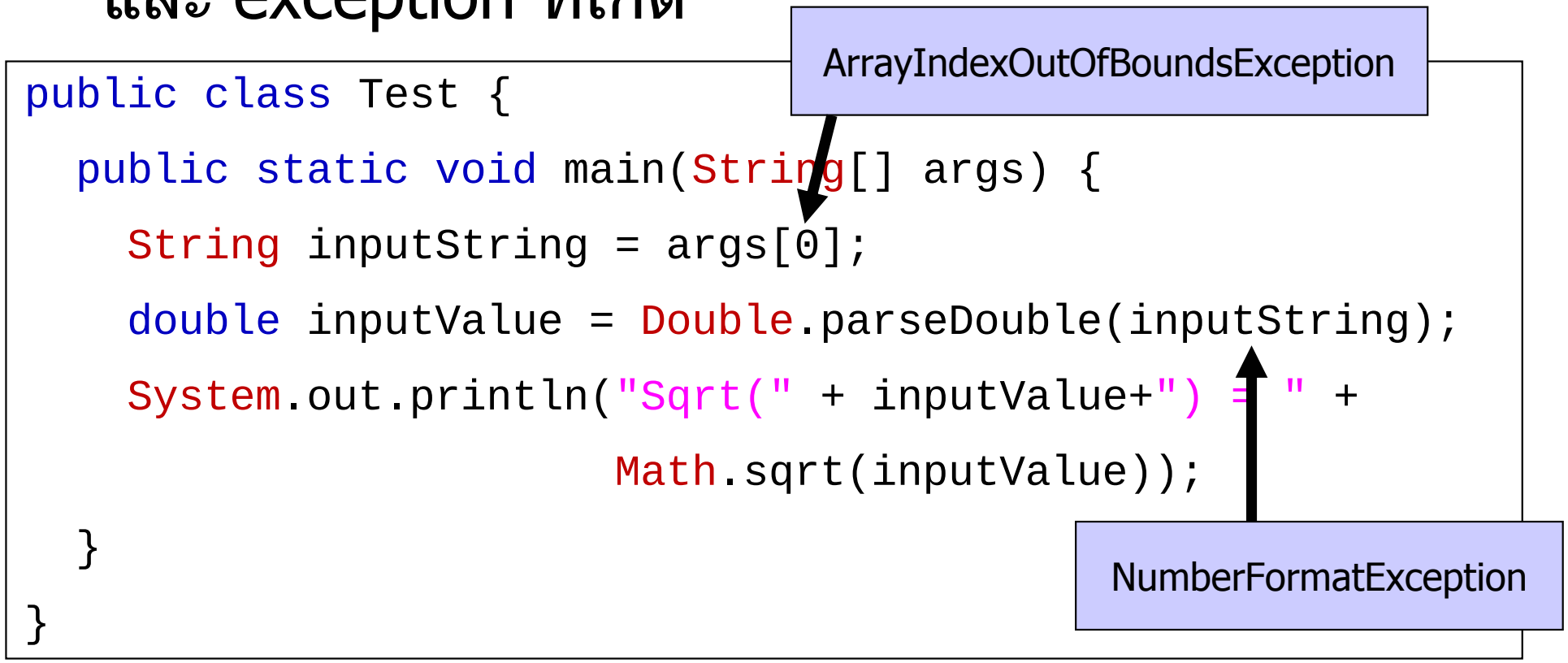
- การทำงานของโปรแกรมมีโอกาสเกิดความผิดพลาดหรือพบสิ่งผิดปกติ จาก
  - คำสั่งต่างๆ ของจาวา
    - หน่วยความจำหมด,
    - หารจำนวนเต็มด้วย 0
    - เปลี่ยนประเภทข้อมูลไม่ได้
    - อ้างอิง index ของอาเรย์นอกช่วงที่จองไว้
    - ...
  - เมท็อดที่เรียกใช้
    - สภาวะของออปเจกต์ผิดปกติ
    - เปิดแฟ้มไม่สำเร็จ
    - พารามิเตอร์มีค่าไม่เป็นไปตามที่กำหนด
    - ...

JVM เป็นผู้ตรวจสอบและแจ้ง error หรือ exception

การตรวจสอบและแจ้ง exception เกิดขึ้นในเมท็อด

การแจ้งความผิดพลาดและสิ่งผิดปกติ

- ใช้การโยน (throw) ออปเจกต์ที่แทน error และ exception ที่เกิด
- ชื่อ class ของออปเจกต์แทนประเภทของ error และ exception ที่เกิด



Exceptions ที่เกิดขึ้นจากระบบจาวา

- การทำงานมีโอกาสผิดปกติ (Exception)

```
// ArithmeticException
int a = 4, b = 0;
int c = a / b;
```

```
// ArrayIndexOutOfBoundsException
int [] a = int[40];
a[40] = 20;
```

```
// StackOverflowError
void Jeng() {
 Jeng();
}
```

```
// ClassCastException
Object a = new Rectangle();
System.out.println((String) a);
```

```
// NegativeArraySizeException
int n = -100;
Vector [] a = new Vector[n];
```

```
// ArrayStoreException
Object [] a = new String[5];
a[0] = new Rectangle();
```

```
 // FileNotFoundException
BufferedReader in = new BufferedReader(
                        new FileReader("data.in"));
String txt = in.readLine();
```

เมื่อเขียนเมท็อดที่อาจแจ้งความผิดปกติ

- เมื่อพบความผิดปกติภายในเมท็อด และต้องการแจ้งให้ผู้เรียกทราบ
  - สร้างออบเจกต์ของคลาสที่แทนความผิดปกติที่เกิดขึ้น
  - throw ออบเจกต์นั้นให้ผู้เรียกรับทราบ
  - เขียนบอกไว้ที่หัวเมท็อดด้วยว่า เมท็อดนี้อาจ throws

```
/**
 * Find real roots of a quadratic equation of the form
 * a*x^2 + b*x + c.
 * @return a two double-element array containing the two roots
 * @throws IllegalArgumentException when there's an imaginary root
 */
public double[] qRoot(double a, double b, double c)
                  throws IllegalArgumentException {
  double t = b * b - 4 * a * c;
  if (t < 0) throw new IllegalArgumentException("imaginary");
  ...
}
```

Example : Account

```
class Account {
  protected long balance;
  public void withdraw(long amount) {
    balance -= amount;
  }
  public void deposit(long amount) {
    balance += amount;
  }
  public long getBalance() {
    return balance;
  }
}
```

ถ้าถอนเป็นจำนวนเงินที่มากกว่าในบัญชี จะทำอย่างไร ?

ถ้าไม่รู้ว่าจะจัดการกับเหตุการณ์เช่นนี้ ก็โยนในผู้เรียก withdraw ทราบ

Be Specific

```
class Account {
  protected long balance;
  public void withdraw(long amount)
                throw IOException {
    if (balance < amount) throw new IOException();
    balance -= amount;
  }
  ...
}
```

ไม่สื่อความหมาย

```
public void withdraw(long amount) throw Exception {
  if (balance < amount) throw new Exception();
  balance -= amount;
}
```

ความหมายกว้างไป

```
public void withdraw(long amount)
              throw InsufficientFundException {
  if (balance < amount)
    throw new InsufficientFundException();
  balance -= amount;
}
```

ชื่อสื่อความหมาย

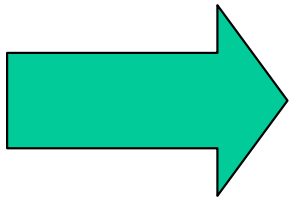
เมื่อใช้เมท็อดที่อาจแจ้งความผิดปกติ

- ขอจัดการ โดยการใช้คำสั่ง try-catch
  - "ดัก" สิ่งผิดปกติ เพื่อจัดการ
  - "ดัก" สิ่งผิดปกติ เพื่อเปลี่ยนประเภทแล้วโยนต่อ
- ไม่ขอจัดการก็โยนต่อ โดยการประกาศไว้ที่หัวเมท็อด

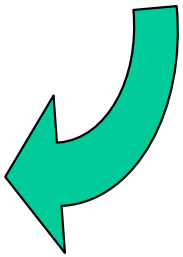
```
try {

  statement
  statement
    ...
  statement

} catch( Exception1 varName ) {
  statement
  ...
  statement
}
```



ถ้าข้อความสั่งใดโยนออบเจกต์ของคลาส Exception1 การทำงานจะกระโดดไปทำงานต่อที่ catch block



ตัวแปรนี้ก็คือตัวอ้างอิงออบเจกต์ที่ถูกโยนออกมาเนื่องจากเกิดสิ่งผิดปกติ

Example : Account

```
public static void main(String[] arg) {
  Account acc = new Account();
  acc.deposit(100);
  acc.withdraw(120);
  System.out.println(acc.getBalance());
}
```

compilation error : ใช้เมท็อดที่อาจโยน exception ก็ต้อง catch หรือไม่ก็โยนต่อ

```
public static void main(String[] arg) {
  Account acc = new Account();
  try {
    acc.deposit(100);
    acc.withdraw(120);
    System.out.println(acc.getBalance());
  } catch (InsufficientFundException e) {
    System.out.println("I won't let you withdraw");
  }
}
```

มีได้หลาย catch blocks

```
try {
  statement
    ...
  statement

} catch( Exception1 varName ) {
  statement
    ...
  statement
} catch( Exception2 varName ) {
  statement
    ...
  statement
} catch( Exception3 varName ) {
  statement
    ...
  statement
}
```

การทำงานจะกระโดดไปทำงานต่อที่ catch block ใด ก็ขึ้นกับว่าออบเจกต์ที่ถูกโยนมาตรงกับคลาสของ catch block ใด

ตัวอย่าง

```
public class Sqrt {
  public static void main(String[] args) {
    try {
      String inputString = args[0];
      double inputValue = Double.parseDouble(inputString);
      System.out.println("Sqrt(" + inputValue+") = " +
                         Math.sqrt(inputValue));
    } catch (IndexOutOfBoundsException e) {
      System.err.println("no number is specified");
    } catch (NumberFormatException e) {
      System.err.println("the argument is not a number");
    }
  }
}
```

normal processing

exception handling

ถ้าไม่จัดการ exception ก็สามารถ “โยน” ต่อ

```
void transfer(Account accFrom, long amount)
                throw InsufficientFundException {
  try {
    accFrom.withdraw(amount);
    this.deposit(amount);
  } catch (InsufficientFundException e) {
    throw e;
  }
}
```

```
void transfer(Account accFrom, long amount)
                throw InsufficientFundException {
  accFrom.withdraw(amount);
  this.deposit(amount);
}
```

สามารถเปลี่ยนประเภท ก่อน “โยน” ต่อ

```
void transfer(Account accFrom, long amount)
                  throw TransferFailException {
  try {
    accFrom.withdraw(amount);
    this.deposit(amount);

  } catch (InsufficientFundException e) {

    throw new TransferFailException();

  }
}
```

Finally block

- เป็น block ที่มาทำงานเสมอไม่ว่าจะเกิด exception หรือไม่ก็ตาม

```
public void method() throws Exception1, Exception2 {
  try {
    ...
  } finally {
    ...
  }
}
```

```
try {
   ...
} catch( Exception1 varName ) {
   ...
} catch( Exception2 varName ) {
   ...
} finally {
   ...
}
```

try-finally

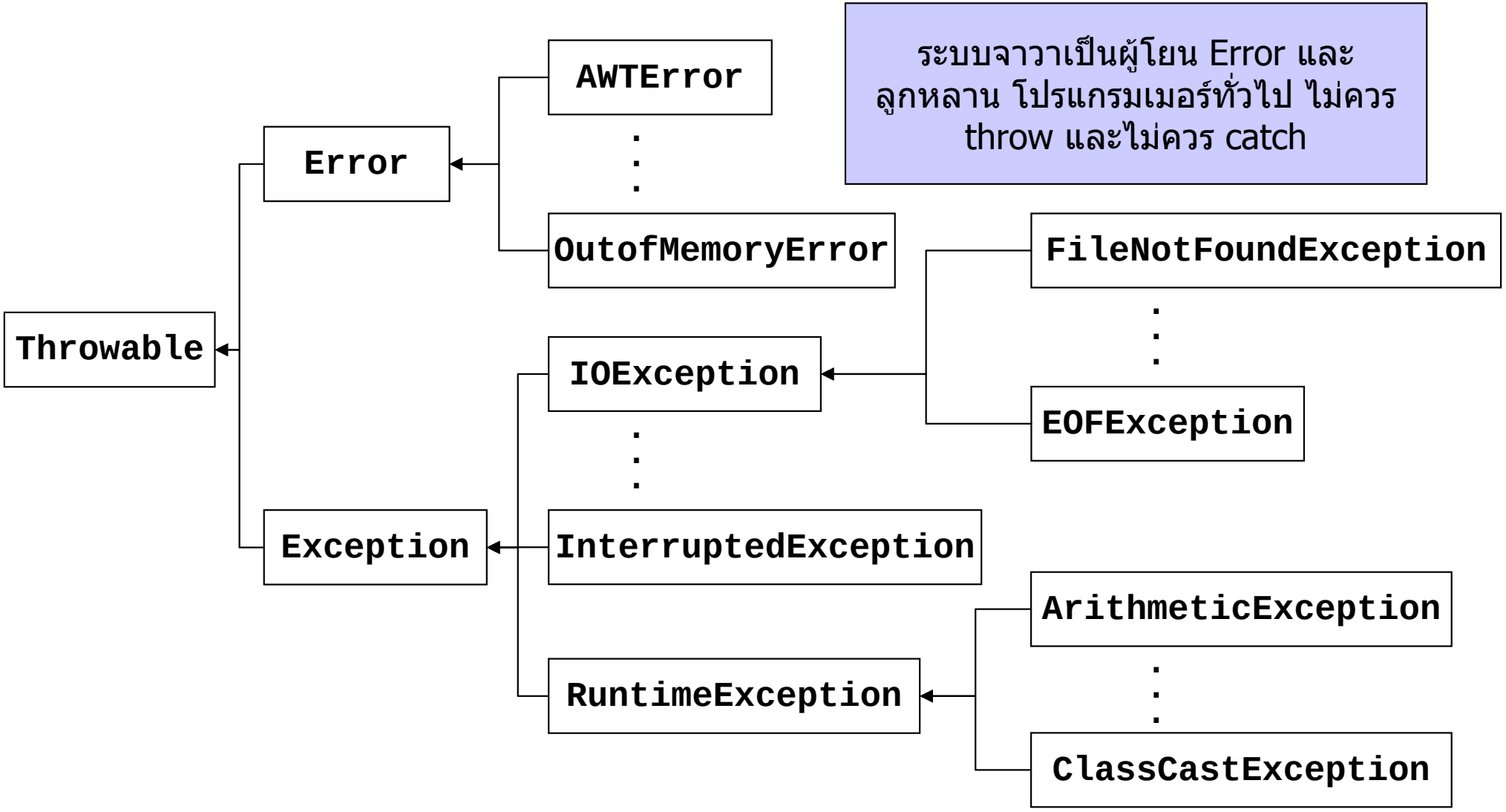
```
public boolean searchFor(String file, String word)
                throws StreamException {
  Stream input = null;
  try {

    input = new Stream(file);
    while (!input.eof())
      if (input.next().equals(word)) return true;
    return false;

  } finally {
    if (input != null) input.close();
  }
}
```

คลาสอะไรที่สร้างออปเจกต์แล้วโยนได้ ?

- โยนได้เฉพาะออปเจกต์ที่ “เป็น” Throwable



Checked & Unchecked Exceptions

- จาวาแบ่ง exceptions เป็น
  - Checked
    - ถ้าในเมท็อดมีการโยนแบบ checked ต้องประกาศที่หัวเมท็อด
    - ถ้าเรียกใช้เมท็อดที่โยนแบบ checked ต้อง catch หรือโยนต่อ
  - Unchecked
    - ถ้าในเมท็อดมีการโยนแบบ unchecked ไม่ต้องประกาศที่หัวเมท็อด
    - ถ้าเรียกใช้เมท็อดที่โยนแบบ unchecked ไม่ต้อง catch หรือโยนต่อ
- compiler จะจู้จี้กับ checked exceptions
- RuntimeException และลูกหลานเป็น unchecked

จาวาเป็นภาษาเดียวที่ใช้ checked exception อย่างจริง ๆ จัง ๆ

Error และลูกหลานก็ถือว่าเป็นแบบ unchecked

Checked Exceptions

- มักใช้กับสิ่งผิดปกติที่มีสาเหตุมาจากภายนอกระบบ
  - หาแฟ้มไม่พบ, ผู้ใช้กรอกข้อมูลผิด, เครื่องแม่ข่ายติดต่อไม่ได้, เครื่องพิมพ์ไม่พร้อม, ...
- checked exception บีบให้ผู้ใช้เมท็อดต้อง catch หรือไม่ก็ต้องโยนต่อ

```
void readProductCode() {
  FileInputStream f = new FileInputStream("code.txt");
  ...
}
```

```
void readProductCode() throws FileNotFoundException {
  FileInputStream f = new FileInputStream("code.txt");
  ...
```

```
void readProductCode() {
  try {
    FileInputStream f = new FileInputStream("code.txt");
    ...
  } catch( FileNotFoundException e ) { ... }
```

Unchecked Exceptions

- มักใช้แสดงสิ่งผิดปกติที่เกิดจากการไม่ปฏิบัติตาม semantic contract เช่น
  - IllegalArgumentException ใช้ในกรณีที่ argument ที่ส่งมาไม่เป็นไปตามที่สัญญากันไว้ใน spec.
    - input ห้ามติดลบ, input ห้ามมีเกิน 4 หลัก, ...
  - IllegalStateException ใช้ในกรณีที่มีการเรียกเมท็อดตอนที่ไม่ควรถูกเรียก
    - ห้ามเรียก exit ก่อน enter, ห้ามเรียก remove ตอน empty, ...
  - NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException, ...
- มักใช้กับ public method
- ถ้าผิด contract ของ private method ควรใช้ assert
- เมื่อเกิด unchecked exception มักหมายถึง bugs

Unchecked Exceptions

```
 /**
   * Returns the element at the specified index in this list.
   *
   * @param i index of element to return.
   * @return the element at the specified index in this list.
   * @throws IndexOutOfBoundsException if out of range
   */
  public void get(int i) {
    if (i < 0 || i >= size)
      throw new IndexOutOfBoundsException("i = " + i);
    ...
  }
}
```

```
List list = new ArrayList();
...
for (int i = 0; i <= list.size(); i++) {
  process(list.get(i));
}
```

compiler ไม่ฟ้องว่าต้อง catch IndexOutOfBoundsException

การสร้าง exception แบบใหม่

- แบบ unchecked ก็ extends จาก RuntimeException
- แบบ checked ก็ extends จาก Exception

```
public class NewUnchecked extends RuntimeException {
  public NewUnchecked () {
    super();
  }
  public NewUnchecked(String s) {
    super(s);
  }
}
```

```
public class NewChecked extends Exception {
  public NewChecked () {
    super();
  }
  public NewChecked(String s) {
    super(s);
  }
}
```

Invoking Methods on Throwable Objects

- เมื่อ catch exception object แล้ว สามารถเรียกใช้เมท็อดของออปเจกต์ได้ (ใน Throwable) เช่น
  - String getMessage( )
    คืน message ที่ผู้สร้าง exception object ส่งให้ constructor
  - void printStackTrace( )
    แสดง stack trace ของการเรียกเมท็อดต่างๆ ณ จุดที่เกิด exception ออกทาง System.err

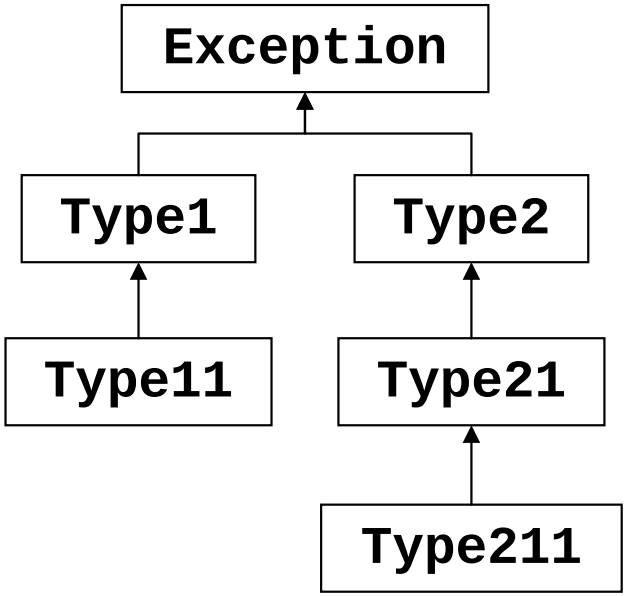
getMessage and printStackTrace

```
01: public class A {
02:   public static void main(String[] args) {
03:     try {
04:       a();
05:     } catch (IllegalStateException e) {
06:       System.out.println(e.getMessage());
07:       e.printStackTrace();
08:     }
09:   }
10:   public static void a() { b(); }
11:   public static void b() { c(); }
12:   public static void c() {
13:     throw new IllegalStateException("Just a test");
14:   }
15: }
```

```
Just a test
java.lang.IllegalStateException: Just a test
        at A.c(A.java:13)
        at A.b(A.java:11)
        at A.a(A.java:10)
        at A.main(A.java:4)
```

เรื่องจุกจิก : ลำดับของ catch clauses

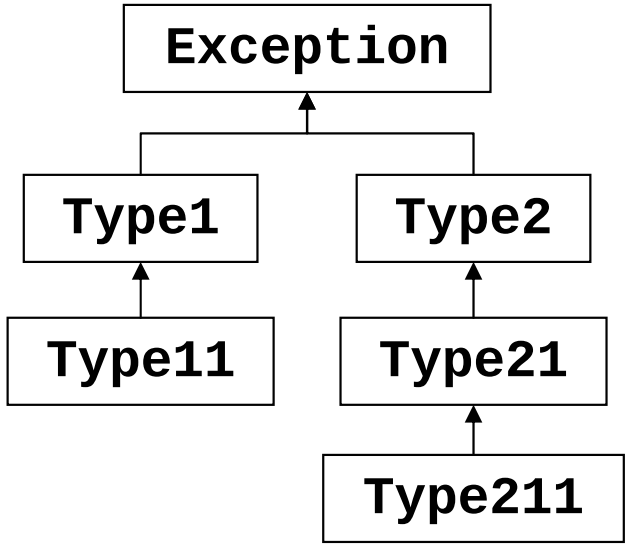
- หนึ่ง try มีหลาย catch blocks ได้
- ถ้ามี exception ที่มีสายสัมพันธ์กัน ต้อง catch ลูกหลานก่อนบรรพบุรุษ




```
try {
   ...
} catch (Type211 e) {
   ...
} catch (Type21 e) {
   ...
} catch (Type11 e) {
   ...
} catch (Type2 e) {
   ...
} catch (Type1 e) {
   ...
} catch (Exception e) {
   ...
}
```

เรื่องจุกจิก : overriding

- เมท็อดของ subclass ที่ overriding ของ superclass ห้ามโยน checked exception ที่ไม่ "เป็น" แบบเดียวกับที่ superclass โยน




```
class A {
  void a() throws Type21 {...}
}
```

```
class B extends A {
  void a() throws Type211 {...}
}
```

```
class C extends A {
  void a() throws Type21, Type211 {...}
}
```

```
class D extends A {
  void a() throws Type1 {...}
}
```

```
class E extends A {
  void a() throws Type2 {...}
}
```

Exception Guidelines

- ใช้ checked exception กับกรณีที่น่าจะจัดการได้
- ใช้ run-time exception กับกรณีที่น่าจะเป็น bugs
- อย่าใช้ exception กับเหตุการณ์ปกติที่ต้องเกิด
- ใช้ exception มาตรฐานที่มีอยู่
  - IllegalArgumentException, IllegalStateException, ...
- throw early, catch late

```
int s = 0;
try {
  int i = 0;
  while (true) s += a[i++];
} catch (ArrayIndexOutOfBoundsException e) {
}
```

Exception Guidelines

- ถ้าไม่จัดการ ให้โยนต่อ อย่า “กลืน”

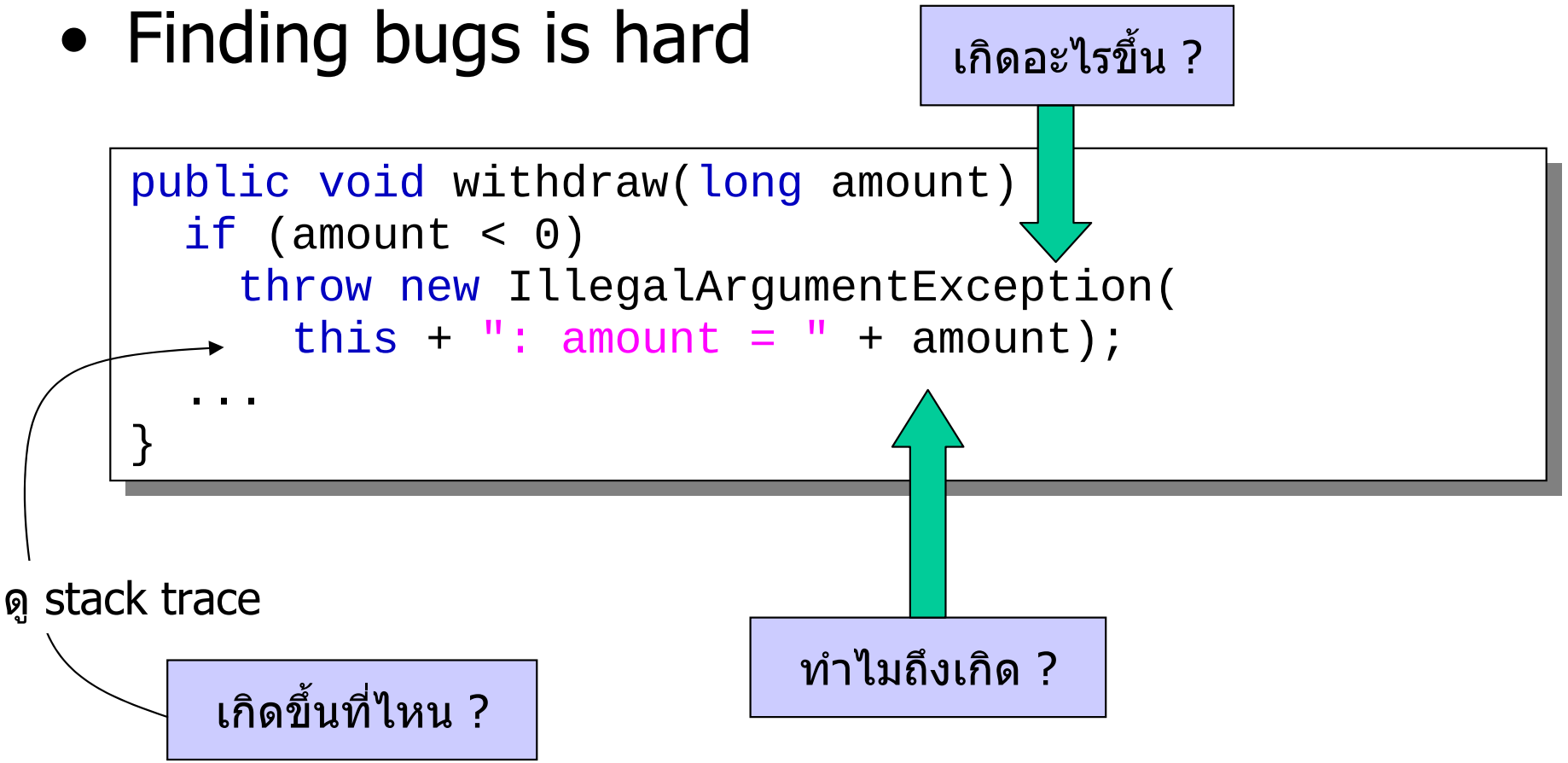
```
void a() {
  try {
    someProcessing();
  } catch (IOException e) {
    handleException();
  }
}
```

```
class A {
  void a() throws IOException {
    someProcessing();
  }
}
```

```
void a() {
  try {
    someProcessing();
  } catch (IOException e) { }
}
```

Bugs

- Fixing bugs is easy
- Finding bugs is hard



Assertion

```
private void heatReactor(int temp) {
  reactor.setHeat(temp);
  ...
}
```

มั่นใจมาก private method เขียนเอง
รับรองว่า temp ที่ส่งมาให้ไม่ร้อนเกินแน่ๆ

```
private void heatReactor(int temp) {
  assert 200 < temp && temp < 1000 : "too hot " + temp;
  reactor.setHeat(temp);
  ...
}
```

กันไว้ดีกว่าแก้ ตรวจสอบให้แน่ใจ ถ้าอยู่นอกช่วงก็ถือว่าเป็น bug

Assertion

- คำสั่งที่ตามด้วย boolean expression ที่โปรแกรมเมอร์มั่นใจว่าต้องเป็นจริง
- ถ้าเกิดเท็จขณะทำงาน จะเกิด AssertionError
- disable การตรวจสอบได้
- โดยทั่วไป enable ตอนพัฒนาและทดสอบ เมื่อมั่นใจว่าปลอดภัย ก็ disable ตอนใช้จริง

```
javac –source 1.4 Test.java

java –ea Test
```

ใช้ได้เฉพาะรุ่น 1.4 เป็นต้นไป

Assertion : Syntax

```
assert booleanExpression;

assert booleanExpression : expression;
```

String ที่แทนข้อความแสดง
รายละเอียดของความผิดพลาด

```
private void heatReactor(int temp) {
  assert 200 < temp && temp < 1000 : "too hot " + temp;
  reactor.setHeat(temp);
  ...
}
```

Assertion : มีไปทำไม ?

- เขียนให้คนเข้าใจและให้เครื่องตรวจสอบ ข้อสมมติของโปรแกรมเมอร์
- ช่วยในการหา bug
- เพิ่มความมั่นใจว่าซอฟต์แวร์ที่ใช้งานไม่มี bug
- ดีกว่า exception ตรงที่ disable การตรวจสอบได้ตอน run-time โดยไม่เสียประสิทธิการทำงานเลย

Assertion : จะเขียนไว้ที่ใด

- precondition ของ non-public methods
- postcondition ของ methods ทั่วไป
- แทน comment ในโปรแกรมด้วย assertion
- เพิ่ม assert ใน switch ที่ไม่มี default
- เพิ่ม assert ใน control flow ที่ไม่มีทางไปถึง

Assertion : Preconditions

```
private void heatReactor(int temp) {
  reactor.setHeat(temp);
  ...
}
```

มั่นใจมาก private method เขียนเอง
รับรองว่า temp ที่ส่งมาให้ไม่ร้อนเกินแน่ๆ

```
private void heatReactor(int temp) {
  assert 200 < temp && temp < 1000 : "too hot " + temp;
  reactor.setHeat(temp);
  ...
}
```

กันไว้ดีกว่าแก้ ตรวจสอบให้แน่ใจ ถ้าอยู่นอกช่วงก็ถือว่าเป็น bug

Assertion : Postconditions

```
public void add(Object v) {
  Entry t = root;
  while (t != null) {
    ...
  }
}
```

มั่นใจมาก เราเขียน put เอง ทดสอบหลายครั้งแล้ว ถูกแน่ ๆ

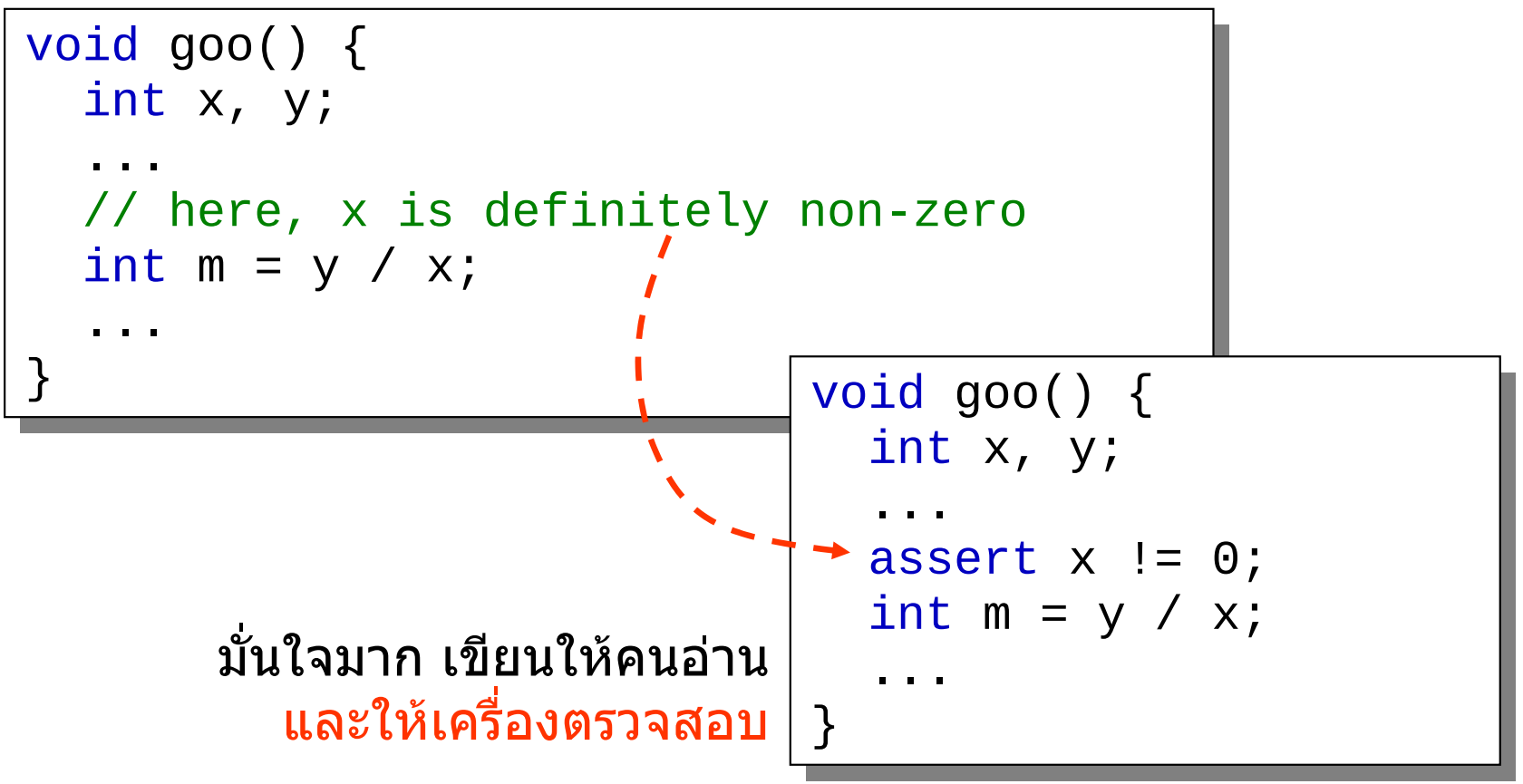
```
public void add(Object v) {
  Entry t = root;
  while (t != null) {
    ...
  }
  assert isBalanced() : "no longer balanced";
}
```

กันไว้ดีกว่าแก้ ตรวจสอบก่อนเสร็จ ว่ายังคงเป็น balanced tree

Assertion : แทน comment

มั่นใจมาก เขียนให้คนอ่านก็พอ

```
void goo() {
  int x, y;
  ...
  // here, x is definitely non-zero
  int m = y / x;
  ...
}
```



มั่นใจมาก เขียนให้คนอ่าน
และให้เครื่องตรวจสอบ

Assertion : ตรงจุดที่มั่นใจว่าไปไม่ถึง

```
Icecream get(int flavor) {
  swith(flavor) {
    case VANILLA :
      ...
    case COCONUT :
      ...
    case LEMON :
      ...
  }
}
```

```
Icecream get(int flavor) {
  swith(flavor) {
    case VANILLA :
      ...
    case COCONUT :
      ...
    case LEMON :
      ...
    default :
      assert false : flavor;
  }
}
```

Assertion : ตรงจุดที่มั่นใจว่าไปไม่ถึง

```
void goo() {
  for (...) {
    ...
    if (...) return;
    ...
  }
  // never reach here
}
```

```
void goo() {
  for (...) {
    ...
    if (...) return;
    ...
  }
  assert false;
}
```

ข้อแนะนำ

- expression ของ assert ต้องไม่มี side effect
- เพิ่ม assert ขณะกำลังเขียน ไม่ใช่เพิ่มหลังเขียนเสร็จ
- โปรย assert ให้ทั่ว ๆ
- ไม่ต้องห่วงเรื่องเวลาการทำงาน เพราะ disable ได้
- Exception ใช้กับสิ่งผิดปกติที่คาดว่ามีสิทธิ์เกิด ถ้าเกิดแล้วมีวิธีจัดการ
- Assertion ใช้กับสิ่งที่ต้องไม่เกิด ถ้าเกิดแสดงว่าผิด
- Exception handling เพิ่ม robustness
- Assertion เพิ่ม reliability