
CSE 143 Java

Errors and Exceptions

Reading: Ch. 18

10/16/2002

(c) University of Washington

07-1

What Can Go Wrong With Programs?

- Programs can have bugs and try to do things they shouldn't.
 - E.g. try to send a message to null
- Users can ask for things that they shouldn't (we can't control the user).
 - E.g. try to withdraw too much money from a bank account
- The environment may not be able to provide some resource that is needed
 - Program runs out of memory or disk space
 - Expected file is not found
 - Extreme network examples:
 - Thousands to millions of tiny sensors
 - Interplanetary Internet

10/16/2002

(c) University of Washington

07-2

Coping Strategies

- Check all user input! (Not doing this has led to many insecurities.)
 - But what should the program do if it's wrong?
- Be able to test whether resources were unavailable.
 - But what should the program do if they weren't?
- Other strategies?

10/16/2002

(c) University of Washington

07-3

Reporting Errors

- If a method cannot complete properly because of some problem, how can it report it to the rest of the program?
- One common approach: return an **error code**
 - A boolean flag: true means OK, false means failure
 - An integer flag: 0 means OK, 1 means error of kind #1, etc.
- Easy to program, in the method that detects the error

```
boolean methodThatMightFail(...) {
    ... if (weirdErrorCondition()) { return false; }
    ... return true;
}
```
- But this is bothersome for callers and unreliable. [Why?]

10/16/2002

(c) University of Washington

07-4

The Original BankAccount

- Part of the original design of the bank account operations:

```
public boolean deposit (double amount) { return this.updateBalance(amount); }
public boolean withdraw(double amount) { return this.updateBalance(-amount); }
```

```
private boolean updateBalance(double amount) {
    if (this.balance + amount < 0) {
        System.out.println("Sorry, you don't have that much money to withdraw.");
        return false;
    } else {
        this.balance = this.balance + amount;
        return true;
    }
}
```

- What's bad about using this boolean error flag (plus a println)?

10/16/2002

(c) University of Washington

07-5

An Alternative: Using Exceptions

- Java (and C++, and many other higher-level languages) include **exceptions** as a better way to report and check for errors

- If something bad happens, can **throw** an exception

- Exceptions are objects of certain classes in Java
- Thrown exceptions abort the throwing method, and its caller, and so on, until a **handler** is found that **catches** the exception

- The handler knows how to cope with the exception



10/16/2002

(c) University of Washington

07-6

Revised BankAccount Methods

```
public void deposit (double amount) { this.updateBalance(amount); }
public void withdraw(double amount) { this.updateBalance(-amount); }
private void updateBalance(double amount) {
    if (this.balance + amount < 0) {
        throw new IllegalArgumentException("insufficient funds");
    } else {
        this.balance = this.balance + amount;
    }
}
```

- All have void return type, not boolean
- Error message and "return false" replaced with throw of new exception object
- Callers can choose to ignore the exception, if they don't know how to cope with it
 - It will be passed on to the caller's caller, and so on, to some caller that can cope

10/16/2002

(c) University of Washington

07-7

Exception Objects

- Exceptions are regular objects in Java
- Exception classes subclasses of the predefined Throwable class
- Some predefined Java exception classes:
 - RuntimeException (a very generic kind of exception)
 - NullPointerException
 - IndexOutOfBoundsException
 - ArithmeticException (e.g. for divide by zero)
 - IllegalArgumentException (for any other kind of bad argument)
- Most exceptions have constructors that take a String argument

10/16/2002

(c) University of Washington

07-8

Throw Statement

- To throw an exception object, use a throw statement
 - Syntax pattern:
`throw <expression whose type is some subclass of Throwable>;`
- Throw is like return: **it ends execution of the containing method**
- But it doesn't just return to the caller, but ends execution of the caller, and its caller, and so on, until a handler is found (explained later), or the whole program is terminated
 - It's bad style for a complete program to die with an unhandled exception

10/16/2002


(c) University of Washington

07-9

Handling Exceptions

- If a caller knows how to cope with an exception, then it can specify an appropriate handler using a **try-catch block**

```
try {
    mySavingsAccount.withdraw(100.00);
    myCheckingAccount.deposit(100.00);
} catch (IllegalArgumentException exn) {
    System.out.println("Transaction failed: " + exn.getMessage());
}
```



- If an exception is thrown anywhere inside the body of the try block, that is an instance of `IllegalArgumentException` or a subclass, then the exception is caught and the catch block is run

10/16/2002

(c) University of Washington

07-10

Try-Catch Blocks: Syntax

- Syntax:

```
try {
    <body, a sequence of statements>
}
catch (<exception type1> <name1>) {
    <handler1, a sequence of statements>
}
catch (<exception type2> <name2>) {
    <handler2, a sequence of statements>
}
...
```

- Can have one or more catch clauses for a single try block

10/16/2002

(c) University of Washington

07-11

Try-Catch Blocks: Semantics

- First evaluate *<body>*
- If no exception thrown during evaluation of *body*, or all exceptions that are thrown are already handled somewhere inside *body*, then we're done with the try-catch block; skip the catch blocks
- Otherwise, if an exception is thrown and not handled, then check each catch block in turn
 - See if the exception is an instance of *<exception type1>*
 - If so, then the exception is caught:
 - Bind *<name1>* to the exception; execute *<handler1>*; skip remaining catch blocks and go to the code after the try-catch block
 - If not, then continue checking with the next catch block (if any)
- If no catch block handles the exception, then continue searching for a handler, e.g. by exiting the containing method and searching the caller for a try-catch block surrounding the call

10/16/2002

(c) University of Washington

07-12

Example

- Implement a robust transferTo method on BankAccount, coping properly with errors that might arise

```
public class BankAccount {  
    ...  
    public void transferTo(BankAccount otherAccount, double amount) {
```