

Objectives

- Exceptions
 - Why Exceptions?
 - Throwing exceptions
 - Catching exceptions
 - Generating our own exception classes

Sept 28, 2011

Sprenkle - CSCI209

1

Review

- How do we specify that a class or a method cannot be subclassed/overridden?
- Compare and contrast abstract classes and interfaces
- When should a class be abstract?
- When should you create/use an interface?
- What is the keyword for defining your class to implement an interface?

Sept 28, 2011

Sprenkle - CSCI209

2

EXCEPTIONS

Sept 28, 2011

Sprenkle - CSCI209

3

Errors

- Programs encounter errors when they run
 - Users may enter data in the wrong form
 - Files may not exist
 - Program code has bugs!*
- When an error occurs, a program should do one of two things:
 - Revert to a stable state and continue
 - Allow the user to save data and then exit the program gracefully

* (Of course, not your programs)

Sept 28, 2011

Sprenkle - CSCI209

4

Java Method Behavior

- Normal/correct case: return specified return type
- Error case: does not return anything, **throws** an Exception
 - An **exception** is an event, which occurs during execution of a program, that disrupts normal flow of program's instructions
 - Exception: object that encapsulates error information

Similar to Python

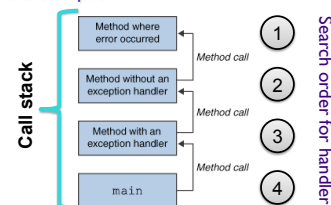
Sept 28, 2011

Sprenkle - CSCI209

5

Handling Exceptions

- JVM's **exception-handling mechanism** searches for an **exception handler**—the error recovery code
 - Exception handler deals with a *particular* exception
 - Searches call stack for a method that can handle (or *catch*) the exception



Sept 28, 2011

Sprenkle - CSCI209

6

Throwable



- All exceptions indirectly derive from **Throwable**
 - Child classes: **Error** and **Exception**
- Important **Throwable** methods
 - **getMessage()**
 - Detailed message about error
 - **printStackTrace()**
 - Prints out where problem occurred and path to reach that point
 - **getStackTrace()**
 - Get the stack in non-text format

Sept 28, 2011

Sprenkle - CSCI209

7

Printing Stack Trace Example

```

java.io.FileNotFoundException: fred.txt
at java.io.FileInputStream.<init>(FileInputStream.java)
at java.io.FileInputStream.<init>(FileInputStream.java)
at ExTest.readMyFile(ExTest.java:19)
at ExTest.main(ExTest.java:7)
  
```

How helpful is this output?
How user friendly is it?

Sept 28, 2011

Sprenkle - CSCI209

8

Printing Stack Trace Example

```

java.io.FileNotFoundException: fred.txt
at java.io.FileInputStream.<init>(FileInputStream.java)
at java.io.FileInputStream.<init>(FileInputStream.java)
at ExTest.readMyFile(ExTest.java:19)
at ExTest.main(ExTest.java:7)
  
```

How helpful is this output?
How user friendly is it?

- Useful for debugging your code
- Generate/display user-friendly errors in finished product
 - Often requires "higher-level code" to handle exception

Sept 28, 2011

Sprenkle - CSCI209

9

Exception Classification: Error

- An internal error
- Strong convention: reserved for JVM
 - JVM-generated when resource exhaustion or an internal problem
 - Example: Out of Memory error
- Program's code should not and can not throw an object of this type
- **Unchecked** exception

When can that happen in Java?

Sept 28, 2011

Sprenkle - CSCI209

10

Exception Classifications

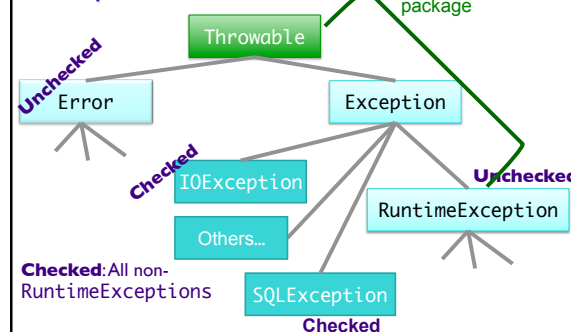
1. **RuntimeException**: something that happens because of a programming error
 - **Unchecked** exception
 - Examples: **ArrayOutOfBoundsException**, **NullPointerException**, **ClassCastException**
2. **Checked** exceptions
 - A well-written application should anticipate and recover from
 - Examples: **IOException**, **SQLException**

Sept 28, 2011

Sprenkle - CSCI209

11

Exception Classification



Sept 28, 2011

Sprenkle - CSCI209

12

Types of Exceptions

Unchecked

- Any exception that derives from `Error` or `RuntimeException`

- Programmer does not create/handle
- Try to make sure that they don't occur
- Often indicates programmer error
 - E.g., precondition violations, not using API correctly

Checked

- Any other exception
 - Programmer creates and handles checked exceptions
 - Compiler-enforced checking
 - Improves *reliability*
- For conditions from which caller can reasonably be expected to recover

Sept 28, 2011

Sprenkle - CSCI209

13

Types of Unchecked Exceptions

- Derived from the class `Error`
 - Any line of code can generate because it is an internal error
 - Don't worry about what to do if this happens
- Derived from the class `RuntimeException`
 - Indicates a bug in the program
 - Fix the bug
 - Examples: `ArrayOutOfBoundsException`, `NullPointerException`, `ClassCastException`

Sept 28, 2011

Sprenkle - CSCI209

14

Checked Exceptions

- Need to be handled by your program
 - Compiler enforced
- Advertise the exceptions that a method throws
 - For each method, tell the compiler:
 - What the method returns
 - What could possibly go wrong
 - Helps users of your interface know what method does and lets them decide how to handle exceptions

Sept 28, 2011

Sprenkle - CSCI209

15

Discussion: Why Checked and Unchecked Exceptions?

- Why do we have exceptions that the compiler doesn't force the programmer to check?
 - Think about examples of unchecked exceptions and when those exceptions can occur

Sept 28, 2011

Sprenkle - CSCI209

16

THROWING EXCEPTIONS

Sept 28, 2011

Sprenkle - CSCI209

17

Methods and Exceptions Example

- `BufferedReader` has method `readLine()`
 - Reads a line from a *stream*, such as a file or network connection
- Method header:

`public String readLine() throws IOException`

Part of "Advertising"
- Interpreting the header: `readLine` will
 - return a `String` (if everything went right)
 - throw an `IOException` (if something went wrong)

Sept 28, 2011

Sprenkle - CSCI209

18

Advertising Checked Exceptions

- Advertising: in Javadoc, document under what conditions each exception is thrown
 - `@throws` tag
- Examples of when your method should advertise the **checked** exceptions that it may throw
 - Your method calls a method that throws a checked exception
 - Your method detects an error in its processing and decides to throw an exception

Sept 28, 2011

Sprenkle - CSCI209

19

Example: Passing an Exception "Up"

```
public String readData(BufferedReader in)
    throws IOException {
    String str1;
    str1 = in.readLine();
    return str1;
}
```

Throws an IOException

- `readData()` calls a method that can throw an `IOException`
- `readLine()` will throw this exception to our method
 - Assuming we don't want to handle the exception, we throw the exception as well
 - Whoever calls `readData` will handle exception

Sept 28, 2011

Sprenkle - CSCI209

20

Throwing An Exception We Created

```
if (month < 1 || month > 12) {
    throw new IllegalArgumentException();
}
```

1. Create a new object of class **`IllegalArgumentException`**
 - Class derived from **`RuntimeException`**
2. **throw** it
 - Method ends at this point
 - Calling method handles exception

Equivalent in Python?

Sept 28, 2011

Sprenkle - CSCI209

21

A More Descriptive Exception

- Four constructors for most Exception classes
 - Default (no parameters)
 - Takes a **String** message
 - Describe the condition that generated this exception more fully
 - 2 more

```
if (month < 1 || month > 12) {
    throw new IllegalArgumentException(
        "Month is not in valid range (1-12)");
}
```

Best messages include all state that could have contributed to the problem

Sept 28, 2011

22

Common Exceptions

Name	Purpose
<code>IllegalArgumentException</code>	When caller passes in inappropriate argument
<code>IllegalStateException</code>	Invocation is illegal because of receiving object's state. (Ex: closing a closed window)

- Both inherit from `RuntimeException`
- May seem like these cover everything but only used for certain kinds of illegal arguments and exceptions
- Not used when
 - A null argument passed in; should be a `NullPointerException`
 - Pass in invalid index for an array; should be an `IndexOutOfBoundsException`

Sept 28, 2011

Sprenkle - CSCI209

23

Factorial Alternatives

```
public static double factorial( int x ) {
    if( x < 0 )
        return 0.0;
    double fact = 1.0;
    while( x > 1 ) {
        fact *= x;
        x--;
    }
    return fact;
}
```

Sept 28, 2011

Sprenkle - CSCI209

24

Factorial Alternatives

Note, no `@throws` clause
Why?

```
public static double factorial( int x ) {
    if( x < 0 )
        throw new IllegalArgumentException("x" +
            "must be >= 0");
    double fact = 1.0;
    while( x > 1 ) {
        fact *= x;
        x--;
    }
    return fact;
}
```

`IllegalArgumentException`:
Thrown to indicate that a method has
been passed an illegal or inappropriate
argument

What are the pros and cons of these approaches?

Sept 28, 2011

Sprengle - CSCI209

25

Rules about `@throws`

- Always report if throw **checked** exceptions
- Report any unchecked exceptions that the caller might reasonably want to catch
 - Exception: `NullPointerException`
 - Allows caller to handle (or not)
 - Document exceptions that are independent of the underlying implementation
- Errors should **not** be documented as they are unpredictable

Sept 28, 2011

Sprengle - CSCI209

26

Goal: Failure Atomicity

- After an object throws an exception, the object should be in a well-defined, usable state
 - A failed method invocation should leave object in state prior to invocation
- Approaches:
 - Check parameters/state before performing operation(s)
 - Do the failure-prone operations first
 - Use recovery code to "rollback" state
 - Apply to temporary object first, then copy over values

Sept 28, 2011

Sprengle - CSCI209

27

Practice

```
public void setBirthday(int month, int day) {
}
```

- How should we implement this method?
- What are some problems we could face?

Sept 28, 2011

Sprengle - CSCI209

28

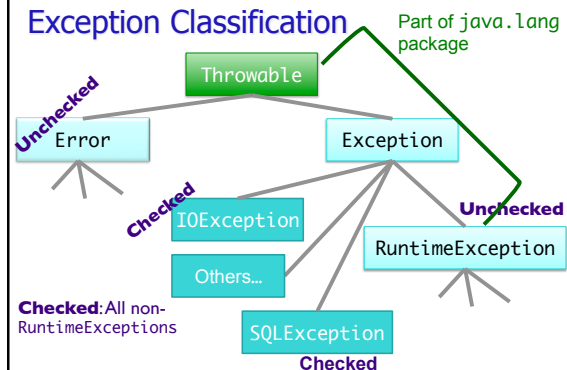
CATCHING EXCEPTIONS

Sept 30, 2011

Sprengle - CSCI209

29

Exception Classification



Sept 28, 2011

Sprengle - CSCI209

30

Catching Exceptions

- After we throw an exception, some part of program needs to *catch* it
 - Knows how to deal with the situation that caused the exception
 - Handles the problem—hopefully gracefully, without exiting

Sept 30, 2011

Sprenkle - CSCI209

31

Try/Catch Block

- The simplest way to catch an exception
- Syntax:

```
try {
    code;
    more code;
}
catch (ExceptionType e) {
    error code for ExceptionType;
}
catch (ExceptionType2 e) {
    error code for ExceptionType2;
}
...
```

Python equivalent?

Sept 30, 2011

Sprenkle - CSCI209

32

Try/Catch Block

```
try {
    code;
    more code;
}
catch (ExceptionType e) {
    error code for
    ExceptionType
}
```

- Code in *try* block runs first
- If *try* block completes without an exception, *catch* block(s) are not executed
- If *try* code generates an exception
 - A *catch* block runs
 - Remaining code in *try* block is not executed
- If an exception of a type other than *ExceptionType* is thrown inside *try* block, method exits immediately*

Sept 30, 2011

Sprenkle - CSCI209

33

Try/Catch Block

```
try {
    code;
    more code;
}
catch (ExceptionType e) {
    error code for
    ExceptionType
}
catch (ExceptionType2 e) {
    error code
    for ExceptionType2
}
```

- You can have more than one *catch* block
 - To handle > 1 type of exception
- If exception is not of type *ExceptionType1*, falls to *ExceptionType2*, and so forth
 - Run the first matching *catch* block

Can catch any exception with **Exception e** but won't have customized messages

Sept 30, 2011

Sprenkle - CSCI209

34

Try/Catch Example

```
public void read(BufferedReader in) {
    try {
        boolean done = false;
        while (!done) {
            String line=in.readLine();
            // above could throw IOException!
            if (line == null)
                done = true;
        }
    }
    catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Prints out stack trace to method call that caused the error

Sept 30, 2011

Sprenkle - CSCI209

35

Try/Catch Example

```
public void read(BufferedReader in) {
    try {
        boolean done = false;
        while (!done) {
            String line=in.readLine();
            // above could throw IOException!
            if (line == null)
                done = true;
        }
    }
    catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

More precise *catch* may help pinpoint error But could result in messier code

Sept 30, 2011

Sprenkle - CSCI209

36

The finally Block

- Optional: add a **finally** block after all **catch** blocks
 - Code in **finally** block **always** runs after code in **try** and/or **catch** blocks
 - After **try** block finishes or, if an exception occurs, after the **catch** block finishes
- Allows you to clean up or do maintenance before method ends (one way or the other)
 - E.g., closing files or database connections

```
try {
    ...
} catch (Exception e) {
    ...
} finally {
    ...
}
```

FinallyTest.java

Sept 30, 2011

Sprenkle - CSCI209

37

Assignment 6

- Due Friday: Practice on Abstract classes and discussion of design decisions, interfaces, packages
- Next Wednesday: Midterm Exam
 - "preparation" document posted today
 - Terminology heavy

Sept 28, 2011

Sprenkle - CSCI209

38

Analysis of equals methods

```
public boolean equals(Object o){
    if(((Birthday) o).getDate() != this.getDate())
        return false;

    if( ((Birthday) o).getMonth() != this.getMonth())
        return false;
    return true;
}
```

```
public boolean equals(Object o) {
    Birthday other = (Birthday) o;
    if (this.month == other.month && this.day ==
        other.day)
        return true;
    else
        return false;
}
```

Sept 28, 2011

Sprenkle - CSCI209

39

Practice: try/catch/finally Blocks

```
try {
    statement1;
    statement2;
} catch (EOFException e) {
    statement3;
    statement4;
} finally {
    statement5;
}
```

- Which statements run if:

- Neither statement1 nor statement2 throws an exception
- statement1 throws an EOFException
- statement2 throws an EOFException
- statement1 throws an IOException

Sept 30, 2011

Sprenkle - CSCI209

40

What to do with a Caught Exception?

- Dump the stack after the exception occurs
 - What else can we do?
- Generally, two options:
 - Catch the exception and recover from it
 - Pass exception up to whoever called it

Sept 30, 2011

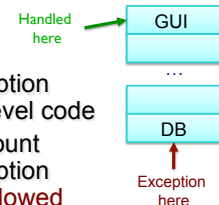
Sprenkle - CSCI209

41

To Throw or Catch?

- Problem: lower-level exception propagated up to higher-level code
- Example: user enters account information and gets exception message "field exceeds allowed length in database"
 - Lost context
 - Lower-level detail polluting higher-level API

Solution: higher-levels should catch lower-level exceptions and throw them in terms of higher-level abstraction



Sept 30, 2011

Sprenkle - CSCI209

42

Exception Translation

```
try {
    // Call lower-level abstraction
}
catch (LowerLevelException ex) {
    // log exception ...
    throw new HigherLevelException(...);
}
```

- Special case: Exception Chaining

- When higher-level exception needs info from lower-level exception

```
try {
    // Call lower-level abstraction
}
catch (LowerLevelException cause) {
    // log exception ...
    throw new HigherLevelException(cause);
}
```

Most standard Exceptions have this constructor

43

Guidelines for Exception Translation

- Try to ensure that lower-level APIs succeed
 - Ex: verify that your parameters satisfy invariants
- Insulate higher-level from lower-level exceptions
 - Handle in some reasonable way
 - Always log problem so admin can check
- If can't do previous two, then use exception translation

Sept 30, 2011

Sprenkle - CSCI209

44

Summary: Methods Throwing Exceptions

- API documentation tells you if a method can throw an exception
 - If so, you **must** handle it
- If your method could possibly throw an exception (by generating it or by calling another method that could), advertise it!
 - If you can't handle every error, that's OK...let whoever is calling you worry about it
 - However, they can only handle the error if you advertise the exceptions you can't deal with

Sept 30, 2011

Sprenkle - CSCI209

45

Programming with Exceptions

- Exception handling is slow
- Use one big **try** block instead of nesting **try-catch** blocks
 - Speeds up Exception Handling
 - Otherwise, code gets too messy
- Don't ignore exceptions (e.g., **catch** block does nothing)
 - Better to pass them along to higher calls

```
try {
    catch O {
    }
    try {
    }
    catch O {
    }
}
try {
    try {
    }
    catch O {
    }
}
try {
    -
    -
    catch O {
    }
```

Sept 30, 2011

Sprenkle - CSCI209

Creating Our Own Exception Class

- Try to reuse an existing exception
 - Match in name as well as semantics
- If you cannot find a predefined Java Exception class that describes your condition, implement a new Exception class!

Sept 30, 2011

Sprenkle - CSCI209

47

Creating Our Own Exception Class

```
public class FileFormatException extends IOException {
    public FileFormatException() {
    }
    public FileFormatException(String message) {
        super(message);
    }
    // other 2 standard constructors...
}
```

What happens in this constructor implicitly?

Is this a checked or unchecked exception?

- Can now throw exceptions of type **FileFormatException**

Sept 30, 2011

Sprenkle - CSCI209

48

Guidelines for Creating Your Own Exception Classes

- Include accessor methods to get more information about the cause of the exception
 - “failure-capture information”
- Checked or unchecked exception?
 - Checked: *forces* API user to handle BUT more difficult to use API
 - Has to handle all checked exceptions
 - Use checked exception if exceptional condition cannot be prevented by proper use of API *and* API user can take a useful action afterward

Sept 30, 2011

Sprenkle - CSCI209

49

Practice: Designing a New Exception Class

- Scenario: When an attempt to make a purchase with a gift card fails because card doesn't have enough money, throw a new exception that you created
- Recall that all Exceptions are Throwable, so they have the methods: `getMessage()`, `printStackTrace()`, `getStackTrace()`

- How would someone else use your class?
- What constructors, additional method(s) may you want to add for your exception class?

Sept 30, 2011

Sprenkle - CSCI209

50

Benefits of Exceptions?

Sept 30, 2011

Sprenkle - CSCI209

51

Benefits of Exceptions

- Force error checking/handling
 - Otherwise, won't compile
 - Does not guarantee “good” exception handling
- Ease debugging
 - Stack trace
- Separates error-handling code from “regular” code
 - Error code is in catch blocks at end
 - Descriptive messages with exceptions
- Propagate methods up call stack
 - Let whoever “cares” about error handle it
- Group and differentiate error types

Sept 30, 2011

Sprenkle - CSCI209

52