Exceptions

- An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
- Many kinds of errors can cause exceptions:
  - Hardware error
  - Programming error: dereferencing `null`
- When such an error occurs within a Java method, the method creates an exception object, which describes the exception, and hands it off to the runtime system, which is responsible for finding code to handle the error.

Exceptions (cont.)

- Where does the runtime system look?
  - The run-time system searches backwards through the chain of method calls, beginning with the method in which the error occurred, until it finds a method that contains an appropriate exception handler.
  - What is an appropriate exception handler?
    * The type of the exception is the same as the type of exception handled by the handler.

Exceptions (cont.)

- What are the advantages?
  - Separates error handling code from regular code.
  - Automatically propagates errors up the chain of method calls.
  - Groups error types and differentiates errors.

Throwing Exceptions

- Example

```
public class Pizza
{
  ...
  public Pizza(double price, AShape shape)
  {
    if (price < 0 || shape == null) {
      throw new IllegalArgumentException(
        "Pizza.Pizza(price, shape): price < 0 or shape == null")
    }
    ...
```

Throwing Exceptions (cont.)

- The argument to `throw` must be a reference to an instance of a subclass of the class `throwable`.
  - `throwable` contains a reference to a descriptive string.
- There are several predefined subclasses of `throwable`.
  - `Exception`
    - `RuntimeException` (extends `Exception`)
  - `Error` *(not usually recovered from)*
- The first type is called a *checked* exception. The compiler verifies that these exceptions are handled or specified.

Throwing Exceptions (cont.)

- The next two types are special: An instance of a subclass of these classes can be thrown from anywhere without specification. Examples of subclass `RuntimeException` are
  - `OutOfMemoryException`
  - `NullPointerException`
  - `NoSuchElementException`
  - `ArrayIndexOutOfBoundsException`

Catching Exceptions

- Example

```
try {
  Pizza badPizza = new Pizza(-4.69, null);
  // Not Reached.
} catch (java.util.IllegalArgumentException e) {
  System.err.println(e);
}
```

- Prints: `java.util.IllegalArgumentException:  Pizza.Pizza(price, shape):  price < 0 or shape == null`

Exceptions Can't Be Ignored...

A method can't ignore exceptions raised by another method that it calls. It must either...

- *catch* the exception or
- *specify* the exception.

Exceptions Can't Be Ignored...(cont.)

- Example

```
method1 {
    try {
        call method2;
    } catch (exception) {
        doErrorProcessing;
    }
}
method2 throws exception {
    call methodThatThrowsException;
}
```

Catching multiple exceptions

- An arbitrary number of `catch` statements can follow the `try` statement.

```
try {
    brokenMethodThrowsExcType12Or3();
    neverCalledMethod();
} catch (ExcType1 e) {
    ...
} catch (ExcType2 e) {
    ...
} catch (ExcType3 e) {
    ...
}
```

Throwing multiple exceptions

- A method can (potentially) throw an arbitrary number of exceptions (but not at once).

```
class Example {
    ...
    void brokenMethodThrowsExcType12Or3()
      throws ExcType1, ExcType2, ExcType3 {
        ...
        try {
            ...
        } catch (ExcType4 e) {
            ...
```

The `finally` statement

- Exceptions can cause control to leave the current method without completing the method's execution. If there is cleanup code at the end of the method, it will never get called.

- The `finally` statement (together with `try`) enables a method to designate code for execution even if an exception occurs.

The `finally` statement (cont.)

- Example

```
try {
    thisMethodThrowsExc();
} finally {
    myCleanup();
}
```

- `myCleanup()` is called regardless of whether an exception is thrown.
- The `finally` statement is **not** a handler. After `myCleanup()` is performed, the exception continues up the call chain in search of a handler.