CS 241 — Introduction to Problem Solving and Programming

Applied Topics

Exceptions

April 13, 2005

Exceptions

We've seen exceptions when our programs crash.

```
public class Exc {
    int[] items;

    // forgot to write a constructor to initialize items

    public static void main(String[] args) {
        Exc e = new Exc();
        e.items[5] = 15;
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException
        at Exc.main(Exc.java:10)
```

Exceptions

```
public class Exc {
    int[] items;
    Exc() { items = new int[5]; }

    public static void main(String[] args) {
        Exc e = new Exc();

        // Should be <, not <=
        for (int i = 0; i <= e.items.length; i++)
            e.items[i] = i;
    }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at Exc.main(Exc.java:12)
```

Exceptions

```
public class Exc {
    int[] items;
    Exc() { items = new int[5]; }

    public static void main(String[] args) {
        Exc e = new Exc();

        for (int i = 0; i < e.items.length; i++)
            e.items[i] = 5 / i;  // what if i = 0?
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Exc.main(Exc.java:11)
```

Exceptions

Exceptions are objects.

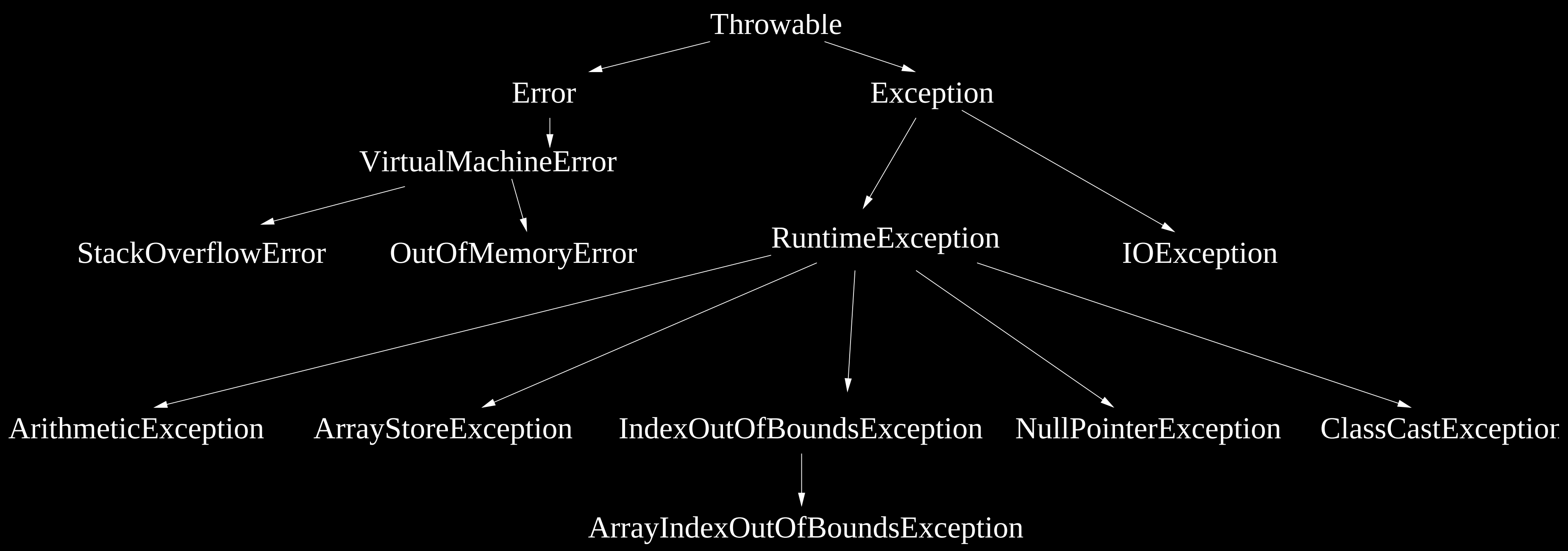
An exception is an instance of the class `Exception` or one of its descendent classes.

An exception is automatically created (the exception class is instantiated) if something goes wrong. The exception is then thrown.

This is also sometimes referred to as raising an exception.

Exception classes

Some classes in the `Exception` family.



`Exception` class

Two methods in particular:

`String getMessage()`— retrieves a message describing the circumstance causing the exception

`void printStackTrace()`— lists the methods that have been invoked but haven't finished when the exception was created

`Class getClass()`— helps you determine what kind of exception this is

But what good does this do? How can I ever use this on an exception object?

First, you have to catch the exception.

Catching exceptions

Enclose the potentially offending line in a try block; follow it with a catch block.

```
public class Exc {
    int[] items;

    public static void main(String[] args) {
        Exc e = new Exc();
        try {
            e.items[4] = 12;
        } catch (Exception ex) {
            System.out.println("I caught a " + ex.getClass());
            System.out.println("It's message is " + ex.getMessage());
            System.out.println("Here's the stack trace:");
            ex.printStackTrace();
        }
        System.out.println("End of program");
    }
}
```

Catching exceptions

```
public class Exc {
    int[] items;

    public static void main(String[] args) {
        Exc e = new Exc();
        try {
            e.items[4] = 12;
        } catch (Exception ex) {
            System.out.println("I caught a " + ex.getClass());
            System.out.println("It's message is " + ex.getMessage());
            System.out.println("Here's the stack trace:");
            ex.printStackTrace();
        }
        System.out.println("End of program");
    }
}
```

Similar to a method declaration, a catch block has a parameter; declare it to be of an exception type and give a name.

Catching exceptions

Inside the catch block, you can refer to the exception by means of the parameter.

```
public class Exc {
    int[] items;

    public static void main(String[] args) {
        Exc e = new Exc();
        try {
            e.items[4] = 12;
        } catch (Exception ex) {
            System.out.println("I caught a " + ex.getClass());
            System.out.println("It's message is " + ex.getMessage());
            System.out.println("Here's the stack trace:");
            ex.printStackTrace();
        }
        System.out.println("End of program");
    }
}
```

Catching exceptions

```
I caught a class java.lang.NullPointerException
It's message is null
Here's the stack trace:
java.lang.NullPointerException
        at Exc.main(Exc.java:9)
End of program
```

Note that the program exits normally.

Uncaught exceptions cause the program to crash.

Catching exceptions

```
public class Exc {
    int[] items;

    public static void main(String[] args) {
        Exc e = new Exc();
        try {
            e.items[4] = 12;
        } catch (Exception ex) {
            System.out.println("I caught a " + ex.getClass());
            System.out.println("It's message is " + ex.getMessage());
            System.out.println("Here's the stack trace:");
            ex.printStackTrace();
        }
        System.out.println("End of program");
    }
}
```

Like any other block statement, you can declare variables, but their scope will be only that block.

Practical use

```
public class Divider {
    public static void main(String[] args) {
        do {
            int dividend = DocsIO.readint("Dividend-> ");
            int divisor = DocsIO.readint("Divisor-> ");
            try {
                System.out.println("Answer: " + (dividend/divisor));
            } catch (ArithmeticException ae) {
                System.out.println("Division by zero");
            }
        } while (DocsIO.readchar("Again (y/n)? ") == 'y');
    }
}
```

Catching exceptions

```
public class Exc {
    int[] items;

    public static void main(String[] args) {
        Exc e = new Exc();
        try {
            e.items[4] = 12;
        } catch (ArrayIndexOutOfBoundsException ex) {
            System.out.println("I caught a " + ex.getClass());
            System.out.println("It's message is " + ex.getMessage());
            System.out.println("Here's the stack trace:");
            ex.printStackTrace();
        }
        System.out.println("End of program");
    }
}
```

A catch will run only if the exception types match. This will nto catch the `NullPointerException`.

Chaining

You can chain catch blocks:

```
try {
    ...
} catch (ExceptionType1 e1) {
    ...
}
} catch (ExceptionType2 e2) {
    ...
}
} catch (ExceptionType2 e2) {
    ...
}
```

Java will try these blocks in order and will execute only the first (not necessarily the best) match.

Methods and exceptions

If a method does not catch an exception, it throw it down to the calling method, which then can catch it.

```
public class Exc {
    int[] items;

    public static void main(String[] args) {
        Exc e = new Exc();
        try {
            m1(e);
        } catch (Exception ex) {
            System.out.println("I caught a " + ex.getClass());
            System.out.println("It's message is " + ex.getMessage());
            System.out.println("Here's the stack trace:");
            ex.printStackTrace();
        }
        System.out.println("End of program");
    }
    static void m1(Exc e) { m2(e); }
    static void m2(Exc e) { m3(e); }
    static void m3(Exc e) { e.items[4] = 12; }
}
```

Methods and exceptions

The stack trace shows the methods currently being called when the exception was created, not necessarily when it was caught.

```
I caught a class java.lang.NullPointerException
It's message is null
Here's the stack trace:
java.lang.NullPointerException
        at Exc.m3(Exc.java:21)
        at Exc.m2(Exc.java:20)
        at Exc.m1(Exc.java:19)
        at Exc.main(Exc.java:9)
End of program
```

Exceptions and expressions

Just as any expression has a type, any expression has a set of exceptions it could possibly throw:

`x + 5`	{ }
`c.x`	{ `NullPointerException` }
`c[14]`	{ `ArrayIndexOutOfBoundsException`, `NullPointerException` }

A method can be declared to throw an exception; that way, the compiler knows that an invocation of that method is potentially excepting.

```
int m(int x, int y) throws ArithmeticException
```

Runtime Exceptions

Exceptions that are not instances of `RuntimeException` or its descendent classes must be caught or the compiler will reject the program.

That is, if an expression could throw a non-`RuntimeException` exception, it must be enclosed in a catch block (or a method that is declared to throw that kind of exception).

Doing it yourself

Like any other class, you can instantiate exception classes:

```
new Exception()

new ArithmeticException("You can’t divide by zero, moron’’);
```

You also can throw an exception

```
throw new Exception();
```

Doing it yourself

```
public class Divider {
    public static void main(String[] args) {
        do {
            int dividend = DocsIO.readint("Dividend-> ");
            int divisor = DocsIO.readint("Divisor-> ");
            try {
                if (divisor == 0)
                    throw new ArithmeticException();
                System.out.println("Answer: " + (dividend/divisor));
            } catch (ArithmeticException ae) {
                System.out.println("Division by zero");
            }
        } while (DocsIO.readchar("Again (y/n)? ") == 'y');
    }
}
```

Throwing and rethrowing

You can also rethrow an exception.

```
public class Exc {
    int[] items;

    public static void main(String[] args) {
        Exc e = new Exc();
        try {
            m1(e);
        } catch (Exception ex) {
            if (ex instanceof NullPointerException)
                throw ex;
            System.out.println("I caught a " + ex.getClass());
            System.out.println("It's message is " + ex.getMessage());
            System.out.println("Here's the stack trace:");
            ex.printStackTrace();
        }
        System.out.println("End of program");
    }
    static void m1(Exc e) { m2(e); }
    static void m2(Exc e) { m3(e); }
    static void m3(Exc e) { e.items[4] = 12; }
}
```

Doing it yourself

You can also make your own exception classes. Often they can be empty, since everything they need is inherited—the class is used only for identification.

```
public class DepartmentHeadException extends Exception

...

     if (userName.equals("perciante"))
         throw new DepartmentHeadException();
```

Summary

- Exceptions as objects
- Exception classes
- Throwing and catching exceptions
- `try` and `catch` blocks
- Stack traces