6: Exceptions

Program Execution

A *program stack* is used to manage the execution of a program.

A *stack frame* on the program stack corresponds to a method call. Every method call results in the creation of a new stack frame.

When an exception occurs, there are a number of active program elements that could potentially handle that problem.

Exception Propagation

If exceptions aren't handled, they propagate back along each method call in the stack. If a method doesn't catch the exception, its execution is interrupted and its stack frame is removed from the stack.

If an exception makes it all the way to the top-level, the default behaviour is to generate a stack trace at the terminal and terminate execution.

Try-Catch Block

A try block contains code that might throw an exception, and a catch block specifies an exception handler for a specific exception.

Checked Exceptions

You can say that a method throws a particular exception:

```
private static void printSpeed(…) throws IllegalArgumentException {}
```

Note: if declaring an exception, it should be as specific as possible.

To throw an exception manually, you use the `throw` keyword:

```java
if (distance < 0 || time <= 0 ) {
    throw new IllegalArgumentException("Distance and time must be 0 or greater.");
}
```

Problems with Exceptions

It is not always possible to see which exceptions might be thrown from looking at source code.

Exceptions also create many possible exit points for a function. Data can be left in an inconsistent state.

Exception handling also carries a performance penalty – avoid placing a try-catch statement inside a loop.

It can be hard to see whether exception-based code is well-written or poorly-written.

Exceptions in Java

It's recommended that you use exceptions for unlikely, unexpected situations. It's also recommended to only catch exceptions if your code can fix the problem.

Finally

The `finally` block is for cleanup actions that *must* be performed – the block is always executed, even if an exception is thrown.