

# Exception Safety

- Code is said to be "exception safe" if it behaves correctly when exceptions are thrown
  - All our code should be exception safe!
- Code is said to be "exception neutral" if it does not handle any exceptions itself, but allows the caller to handle it
  - Constructors should usually be exception neutral

## Exception Guarantees

- There are three main ways in which we can write exception-safe code
- Basic exception guarantee
  - If an exception is thrown, no resources are leaked
- Strong exception guarantee
  - If an exception is thrown, the program reverts to its previous state
- No-throw guarantee
  - The code does not throw any exceptions

- An operation that uses resources will either succeed or throw an exception
  - If it throws an exception, no resources will be leaked
  - All code that is expected to be able to recover from an error condition should provide the basic guarantee
  - All the code in the C++ standard library conforms to the basic guarantee

- An operation that uses a resource will either succeed or have no effect
  - If it throws an exception, the program will be left in the same state as it was before the error condition
  - The operation has transactional semantics, similar to commit and rollback in databases
  - All iterators or references which were obtained before the operation must remain valid even if the operation fails
  - All STL containers conform to the strong exception guarantee (except for a couple of special cases of insert operations)

## Example of Exception Guarantees

```
stack<int> s;  
s.push(5);           // Populate stack  
auto& t = s.top();   // Get reference to the top element of the stack  
try {  
    s.pop();          // Exception is thrown here for some reason  
} catch (const std::exception& e) {  
    ....              // Exception is handled here  
}  
cout << t << endl;  // Is this safe?
```

- Basic guarantee: no memory has been leaked
- Strong guarantee: t is still a valid reference

- The operation will not throw an exception
  - we need to know that sub-operations will not throw in order to write code that provides the basic or strong guarantees
  - `std::swap()` is very useful when writing exception-safe code. It will not throw an exception
  - None of the built-in functions and operators in the core C++ language throw exceptions, apart from:
    - `new`
    - `dynamic_cast`
    - `throw`

- C++11 introduced the `noexcept` keyword
- This declares that a function will not throw any exceptions

```
void does_not_throw(int input) noexcept {    // I promise not to throw any exceptions  
    // No exceptions thrown here, honest  
}
```

- This is useful information for the caller and helps the compiler optimize the code
- However, the compiler does not check the function for exceptions and will not give an error if the function throws



## throw()

- Earlier versions of C++ used throw() with no arguments instead of noexcept

```
void does_not_throw(int input) throw() {    // I promise not to throw any exceptions  
    // No exceptions thrown here, honest  
}
```

- Again, this was not enforced by the compiler

## throw() with arguments

- throw() could also take arguments
  - These listed the exceptions that the function could throw

```
void get_data() throw(NetworkConnectionError, DataError) {  
    // Could throw NetworkConnectionError exception or DataError  
}
```
- The list of exceptions was not checked by the compiler
  - Only useful if programmers are diligent about listing exceptions!
- throw() has mostly been removed from the language, but the version with no arguments can still be used

- Java has checked and unchecked ("runtime") exceptions
  - If a function can throw a checked exception, every caller of that function must use a try/catch block which handles that exception when calling that function
  - If a function can throw an unchecked exception, the caller is not required to handle that exception. The exception can be handled by the caller's caller or the caller's caller's caller...
- C++ does not have checked exceptions
  - All exceptions in C++ work the same way, which is similar to unchecked exceptions in Java

## Summary

- Code is exception-safe if it behaves correctly when an exception is thrown
- Basic exception guarantee: If an exception is thrown, no resources will be leaked
- Strong exception guarantee: An operation that uses a resource will either succeed or have no effect
- No-throw guarantee: No exceptions are thrown
- In C++11, the `noexcept` keyword shows that a function does not throw exceptions