Introduction to Exceptions

An **exception** is an abnormal condition that occurs during the execution of a program. For example, divisions by zero, accessing an invalid array index, or trying to convert a letter to a number are instances of exceptions. The concept is essentially the same in C++ as in Java, with a few syntax differences. Exceptions are used more commonly in Java than C++.

As an example, consider this toy example, which withdraws money from a pretend bank account. This version uses no exceptions. It just uses an if statement to check if we try to withdraw more money than exists in the account.

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main()
{
  int account = 100;  // Pretend we have $100 in our account
  int withdrawal;

  cout << "Enter an amount to withdraw." << endl;
  cin >> withdrawal;
  if (account - withdrawal < 0)
  {
        cout << "Not enough money!" << endl;
  }
  else
  {
        account -= withdrawal;
        cout << "You got " << withdrawal << " dollars." << endl;
        cout << "Your account has $: " << account;
  }
}
```

Nothing too exciting to see here – so let's go straight to an exception version of this program. In this case we can throw an exception when we try to withdraw more money than is in the account. We catch the exception and handle it somehow. In this case, we just print out the error message. The code is somewhat more complicated but normally we won't use exceptions quite this way. We can throw an exception of any data type and we can catch any data type, so if we wanted we could throw a string or double.

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main()
{
  int account = 100;  // Pretend we have $100 in our account
  int withdrawal;
  cout << "Enter an amount to withdraw." << endl;
  try
  {
   cin >> withdrawal;
   if (account - withdrawal < 0)
        throw 1;
   account -= withdrawal;
   cout << "You got " << withdrawal << " dollars." << endl;
   cout << "Your account has $: " << account;
  }
  catch (int e)
  {
        cout << "Exception code is " << e << " withdrawal problem"
             << endl;
  }
}
```

The advantage of this approach is we can now throw multiple exception codes for different types of exceptions. For example, we could throw the string "Withdrawal exceeds max" if we want to have an exception for trying to withdraw over 1000:

```
int main()
{
  int account = 100;  // Pretend we have $100 in our account
  int withdrawal;

  cout << "Enter an amount to withdraw." << endl;
  try
  {
   cin >> withdrawal;
   if (withdrawal > 1000)
        throw "Withdrawal exceeds max";
   if (account - withdrawal < 0)
        throw 1;
   account -= withdrawal;
   cout << "You got " << withdrawal << " dollars." << endl;
   cout << "Your account has $: " << account;
  }
  catch (int e)
  {
        cout << "Exception code is " << e << " withdrawal problem"
             << endl;
  }
  catch (char const *s)
  {
        cout << s << endl;
  }
}
```

I used "char const *s" as the type for the second catch, since throwing " " by default in C++ is a c-style string, not a STL string.

More typically, the reason for the try-throw-catch is because the code that detects the exception doesn't know how to handle it. In the above examples, it is probably better written using a regular if-statement because we know how to handle the different situations that might arise.

When wouldn't we know how to handle it? Consider a function that does the withdrawal calculation. Maybe this calculation is done on the bank server or somewhere that doesn't have access to the console to print error messages to the user. The typical solution is to return some error code:

```
int withdraw(int account, int withdrawal)
{
   if (account - withdrawal < 0)
   {
        // What do we do if we can't use output to the user here?
        // Maybe return an error code
        return -1;
   }
   // Return new balance
   return account - withdrawal;
}

int main()
{
  int account = 100;  // Pretend we have $100 in our account
  int withdrawal;

  cout << "Enter an amount to withdraw." << endl;
  cin >> withdrawal;
  account = withdraw(account, withdrawal);
  if (account > -1)
  {
   cout << "You got " << withdrawal << " dollars." << endl;
   cout << "Your account has $: " << account;
  }
  else
    cout << "Withdrawal error." << endl;
}
```

This works (there are perhaps better ways to do this, like passing account by reference) but what if we want to allow negative account balances? Then there isn't a way to distinguish the return value as an error code from the actual account balance. It also relies on the user checking the error code.

An alternate solution is for the function to throw an exception and abort. Essentially the function is saying "I don't know how to handle this issue, but it happened, so whoever called me, you get to handle it." To do this, we normally define our own exception class. It can be trivial. Here are two, one for exceeding the maximum withdrawal amount, and the other for exceeding our balance:

```
class ExceedsMax
{ };
class ExceedsBalance
{
 public:
   ExceedsBalance() { message = "Need mo money.";};
   ExceedsBalance(string s) { message=s; };
   string message;
};
```

Here is how we modify the withdraw function to throw these exceptions. We have to tack on the exceptions thrown at the end of the function header:

```
int withdraw(int account, int withdrawal)
            throw (ExceedsMax,ExceedsBalance)
{
   if (withdrawal > 1000)
        throw ExceedsMax();
   if (account - withdrawal < 0)
        throw ExceedsBalance("Not enough money in the account.");
   // Return new balance
   return account - withdrawal;
}
```

If we leave main unchanged, everything is happy if there is no exception:

```
Enter an amount to withdraw.
30
You got 30 dollars.
```

But if we enter an exception condition:

```
Enter an amount to withdraw.
102
terminate called after throwing an instance of 'ExceedsBalance'
```

The program crashes!   If we don’t catch the exception thrown then C++ “catches” it for us, which basically makes the program stop. In this way, throwing exceptions is like the nuclear option. We could return error codes that the programmer might ignore. But the programmer can’t ignore an exception (well, except by writing an empty catch block). To keep the program from crashing we would handle these exceptions in main:

```
int main()
{
  int account = 100;  // Pretend we have $100 in our account
  int withdrawal;

  cout << "Enter an amount to withdraw." << endl;
  cin >> withdrawal;
  try
  {
    account = withdraw(account, withdrawal);
    cout << "You got " << withdrawal << " dollars." << endl;
    cout << "Your account has $: " << account;
  }
  catch (ExceedsBalance e)
  {
    cout << e.message << endl;
  }
  catch (ExceedsMax e)
  {
    cout << "You exceeded the maximum withdrawal amount." << endl;
  }
}
```

Sometimes exceptions are overused – once again they are most appropriate inside a function that isn’t able to handle the exception case itself. In that case, throw an exception and let the calling code handle it.