CS 216

Lecture 10

March 14th, 2014

Administrivia

Midterm exam: 12.5% curve

Midterm grades:
Posting today

Please register your clickers on the source control web site.

PA2 deadline extended until 3/21

PA3 will be posted today

Solutions to PA1 and PA2 will be made available after spring break.

And now, content!

Exception handling

Normally we have return types

And often those return types are used to indicate error

But sometimes we want a bit more…

Enter exception handling.

```
try
{
}
catch(string s)
{
}
catch(...)
{
}
```

try block
one or more
catch blocks

throw keyword

What gets thrown is called an exception

Exceptions can be any type, but often have specific exception classes

If an exception is thrown, it gets caught by the most immediate catch block

And execution jumps to the catch block.

Which can be confusing, but…

You can have nested try/catch structures

And it works through function calls.

Gotcha:
Exceptions do not have implicit type conversion.

Why?

More robust error handling.

It means our return type (or parameters) do not need to encode all possible errors.

Because in general, we're more interested in those values when the call *works*.

It gives the programmer more choice *where* the error is handled.

And really most importantly:

It creates less code to maintain, since you aren't checking every single call for an error return.

Why not?

Overuse can be confusing.

Historically, they have been slow, but with modern compilers this is no longer particularly true.

Concrete example: Database queries

Even more concrete: Disk files

We can handle file problems in one place instead of at every file operation.

Why

Universality

Exception handling

```
try
{
    // work goes here
}
catch(string sError)
{
    cout << "Error caught: " << sError << endl;
}
```

```php
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('Division by zero.');
    }
    else return 1/$x;
}

try {
    echo inverse(5) . "\n";
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Caught exception: ',  $e->getMessage(), "\n";
}

// Continue execution
echo 'Hello World';
?>
```

```java
// File Name : ExcepTest.java
import java.io.*;
public class ExcepTest{

   public static void main(String args[]){
      try{
         int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the block");
   }
}
```

```
- (void)endSheet:(NSWindow *)sheet
{
    BOOL success = [predicateEditorView commitEditing];
    if (success == YES) {

        @try {
            [treeController setValue:[predicateEditorView predicate] forKeyPath:@"selection.predicate"];
        }

        @catch ( NSException *e ) {
            [treeController setValue:nil forKeyPath:@"selection.predicate"];
        }

        @finally {
            [NSApp endSheet:sheet];
        }
    }
}
```

Inheritance

```
class Item : public Entity
{
public:
    Item(); // constructor
    std::string getDescription() const;
    int getWeight() const;
    int getValue() const;
    int getRarity() const;
```

```
public class MountainBike extends Bicycle {

    // the MountainBike subclass adds one field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int startHeight,
                        int startCadence,
                        int startSpeed,
                        int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

```
@interface Square: Rectangle


<strong>Square.h</strong>

1
#import "Rectangle.h"

@interface Square: Rectangle
-(Square*) initWithSize: (int) s;
-(void) setSize: (int) s;
-(int) size;
@end
```

```php
/* using the keyword EXTENDS to make Student inherit from Person */
class Student extends Person
{
     public function __construct($name = 'unknown',
                                          $surname = 'unknown',
                                          $id = 0,
                                          $topic = 'IT')
     {
               //ALWAYS call the parent constructor
               //from a derived class
               parent::__construct($name, $surname);
               $this->id = $id;
               $this->topic = $topic;
     }//end constructor
```