UNIVERSITY OF Hull

Method Libraries

08101 Programming 1
C# Programming

Creative Laziness

- A good programmer is “creatively lazy”
- This means that they will try to avoid writing the same program more than once
- One way to do this is to create your own library of methods that your programs can use

Reading Numbers

- One thing that we need to do a lot is read in numbers
- Each time we read in a number we will have a maximum and minimum value that it can hold
- We will also have a prompt that we want to display for the user

Using a Read Number method

```
Enter the x coordinate (0 to 9) : 10
Number too large
Enter the x coordinate (0 to 9) : six
Invalid number
Enter the x coordinate (0 to 9) : 6
```

- What we want is a user interface like this
- The user types in a value which is rejected if it is out of range

Creating a Method

- We can feed values into a method as parameters and it can return a single value
- The `ReadNumber` method needs to accept three things:
  - The largest number
  - The smallest number
  - The prompt string

The method signature

```
static int ReadNumber ( int min, int max,
                        string prompt )
```

- The description of a method is called the *signature*
- This shows the name of the method, the parameters that it has and the value it returns

Calling the method

```
x = ReadNumber ( 0, 9,
                 "Enter the x coordinate" );
```

- When we call the method we need to give it the three parameters that we specified in the signature
- It returns the integer that it has read

A dummy method

```
static int ReadNumber ( int min, int max,
                        string prompt )
{
    return 0;
}
```

- This is a dummy method
- It doesn't actually read the number, it just returns 0
- It will compile OK, but can't be used

What the method must do

- Start a loop
  - Print the prompt
  - Read the string
  - Convert the string into a number
    - Catch any exceptions that are thrown
  - Check the range of the number entered
    - Break out of the loop if the number is in range

The method loop

```
static int ReadNumber ( int min, int max,
                        string prompt )
{
    while (true)
    {
        // read numbers
        // break with a value in range
    }
}
```

- This is loop we are going to use

Returning a result

```
static int ReadNumber ( int min, int max,
                        string prompt )
{
    int result;
    while (true)
    {
        // read numbers and store in result
        // break with a value in range
    }
    return result;
}
```

Returning a result

```
static int ReadNumber ( int min, int max,
                        string prompt )
{
    int result;
    while (true)
    {
        // read numbers and store in result
        // break with a value in range
    }
    return result;
}
```

Printing a prompt

```
Console.Write( prompt + "(" + min + " to " +
               max + ") : " );
```

```
Enter the x coordinate (0 to 9) :
```

- This write statement will write out the prompt for the user
- It uses the max and min parameters

Reading a number

```
string resultString = Console.ReadLine();
try
{
    result = int.Parse(resultString);
}
catch
{
    Console.WriteLine("Invalid number");
    continue;
}
```

- This continues round the loop if the number is invalid

Using Continue

- The continue keyword does not break out of a loop
  - That is what the break keyword does
- Instead the continue keyword causes the loop to go round again
- That is what we want to do, because we need to get another value from the user

Range Checking

```
if (result > max)
{
    Console.WriteLine("Number too high");
    continue;
}
```

- This tests that the result is not too large
- If the result is greater than the maximum an error message is printed and the loop is repeated

Breaking out of the loop

```
if (result < min)
{
    Console.WriteLine("Number too small");
    continue;
}
break;
```

- This is the test for a value that is too small
- If the code gets past this test it can break out of the loop

Returning a result

```
static int ReadNumber ( int min, int max,
                        string prompt )
{
    int result;
    while (true)
    {
        // read numbers and store in result
        // break with a value in range
    }
    return result;
}
```

Methods and design

- We can create the designs for the methods before we create the system itself
- We can even create dummy methods which can be filled in later
  - These are called “stubs”
- We can also create tests for the methods and test them individually

Methods and Errors

```
x = ReadNumber ( 9, 0, "Enter x" );
```

- There are a number of ways that my method could be upset
  - The minimum could be larger than the maximum
- This would cause my method to look stupid – which I hate

Error handling

- When you design a method you should also plan how the error handling should work for the method
- In the case of this method there are a number of ways it could deal with the problem:
  - Swap the maximum and minimum values
  - Return an out of range value

Swapping max and min

- It would be easy for the method to swap the maximum and minimum around and use them in the "correct" order
- But this is **very** dangerous
  - We are assuming that we know what the mistake was
  - If the programmer had miss-typed the values then our "fix" would make things worse

Returning an invalid value

- Another solution might be for the method to return a value which means "invalid"
  - Perhaps 1 less than the minimum
- But this is **even more** dangerous
  - We are assuming that the user will check for the "invalid" value and then do something sensible when it is entered
  - If they don't their program will misbehave

Throwing an Exception

- In this case I think it is perfectly valid for a method to throw an exception to indicate that the parameters are invalid
- This will make sure that the user of my method knows that they have done something wrong
- They will have to fix the problem

Throwing an exception

```
static int ReadNumber ( int min, int max,
                        string prompt )
{
    if (min >= max)
    {
        throw new Exception ("Invalid range");
    }
    // reset of method here
}
```

Designing error handling

- Whenever you create a method to do something you should consider how that method can fail
- The behaviour of a method when it fails should be designed into your system
- Perhaps the method could return an error code or message

Proper programming

- A programmer will probably spend more time thinking about the error conditions and how a method can fail than they will writing the code that does the work
- This is perfectly sensible
- It also means that programming jobs are more complex than you thought

Summary

- When we have an action that we need to perform in lots of different parts of the program we should create a method
- The method will have a particular signature
- We can design the methods and how they work before we write them
- We also need to consider error handling