More Objects & Generics

Guest Lecturer: Brian Aydemir

January 21, 2008

Annoucements

- Peter-Michael Osera is our new grader and unofficial TA.
- Problem Set 1: Due before class last night(!)
- Problem Set 2
  - Posted today.
  - Due February 3, 11:59pm.
  - PS 2 will be with partners. If you have a partner, mail Jeff with your names by Friday. Jeff will assign groups to anyone he hasn't heard from on Saturday.
  - Start early.
- Jeff will be back next week.

Interfaces declare contracts that a class must follow.

- Interfaces list methods which much a appear in a class.
- Methods may use interface names for argument and result types (bounded polymorphism).
- Classes can implement interfaces in two ways
  - Implicitly (the normal way), interface methods added directly to class and accessed as usual.
  - Explicitly, interface members are declared with special syntax and accessed through casts. Useful in the case where two interfaces declare methods with the same name.

Example: Implicit Interface Implementation

```
interface IWindow {
  void Draw();
}

public class Display: IWindow {
  // Implicit Interface Implementation
  public void Draw(){ Console.Out.WriteLine ("A");}
}

class Runner{
  static void Main(string[] args){
    Display c = new Display();
    d.Draw(); // "A"
} }
```

Multiple interfaces can conflict.

```
interface IWindow {
  // Implementations should print to the screen
  void Draw();
}

interface ICowboy {
  // Implementations should get out a gun
  void Draw();
}

// Trouble!
public class WesternGame: IWindow, ICowboy {...}
```

Example: Explicit Interface Implementation

```
class WesternGame: IWindow, ICowboy {
  // Explicit Interface Implementations
  void IWindow.Draw(){
    Console.Out.WriteLine ("Drawing␣Picture"); }
  void ICowboy.Draw(){
    Console.Out.WriteLine ("Drawing␣Six␣Shooter");}
}

class Runner{
static void Main(string[] args){
  WesternGame w = new WesternGame();

  // Error: w.Draw();
  ((ICowboy) w).Draw(); // "Drawing Picture"
  ((IWindow) w).Draw(); // "Drawing Six Shooter"
}}
```

Casting

```
string x = (string) someObject;
```

- Up-casts:
  - Convert instances of a child class to a parent class or interface.
  - Always succeeds.
- Down-casts:
  - Convert instances of a parent class to a child class.
  - May fail and throw InvalidCastException
  - Use `as` or `is` to check if a cast is safe.
- Generics provide an elegant way to write (for example) collection classes without casting.

Generics allow types (e.g. classes and delegates) to be parameterized by types.

- Provide extra compile-time type information
- Provide opportunities for compiler optimizations.
- Allow the compiler to catch bugs that would otherwise happen at runtime.
- Enhance code readability.
- Reduce need for downcasts (which are expensive and can throw exceptions).

Example: a specialized “option” class.

```
public class IntOption{
  private bool isFull; private int contents;

  public bool isEmpty() { return !isFull; }

  public int GetValue() {
    if (isFull) return contents;

    throw new Exception("GetValue of Empty");
  }

  public IntOption() { isFull = false; }

  public IntOption(int x){
      isFull = true; contents = x; }
}
```

Example: Using the specialized option class.

```
public class Runner{
  public static IntOption div(int x, int y){
    if (y==0)
        return new IntOption();
    else
        return new IntOption(x / y);
  }

  public static void Main(string[] args)
  {
    C.O.WriteLine(div(3,4).isEmpty()); //false
    C.O.WriteLine(div(3,0).isEmpty()); //true
  }
}
```

Example: A generic option class.

```
public class GenOption<T>{
  private bool isFull; private T contents;

  public bool isEmpty() { return !isFull; }

  public T GetValue() {
    if (isFull) return contents;

    throw new Exception("GetValue of Empty");
  }

  public GenOption() { isFull = false; }

  public GenOption(T x){
      isFull = true; contents = x; }
}
```

Example: Using the generic option class.

```
public class Runner{
  public static GenOption<int> div(int x, int y){
    if (y==0)
        return new GenOption<int>();
    else
        return new GenOption<int>(x / y);
  }

  public static void Main(string[] args)
  {
    C.O.WriteLine(div(3,4).isEmpty()); //false
    C.O.WriteLine(div(3,0).isEmpty()); //true
  }
}
```

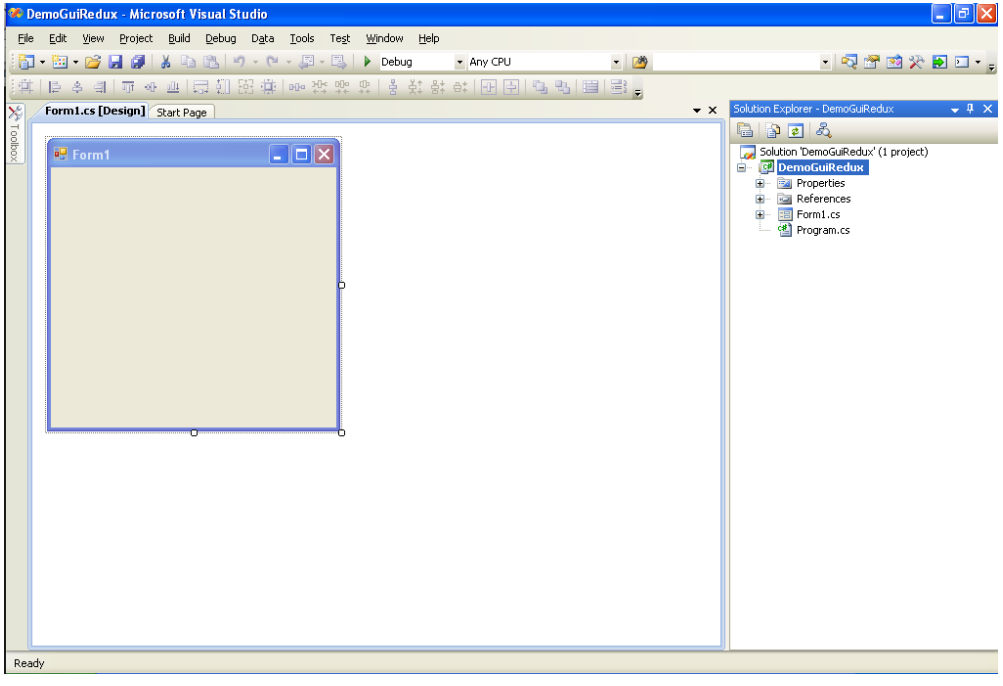
Generics vs. Generics vs. Templates

- C++ Templates
  - Template expansion is static: each template instantiation creates a new compile-time class.
  - Templates can't live in compiled libraries—only headers.
  - Templates expansion = Turing complete programming language(!)
- Java Generics
  - Similar semantics to C#
  - Implemented by type erasure; no runtime support in JVM
  - Poor support for reflection
  - Legacy code can break apparent type guarantees for generic objects.
- C#Generics
  - CLR (.Net virtual machine) has support for generics
  - Generics can be specialized to used native types at runtime
  - Type parameters preserved at runtime, and can be queried by reflection.

A step-by-step guide to building a simple GUI

If you missed the in-class demo, try following these notes to build a simple gui using Visual Studio and Windows Forms.

- Start a “Windows Forms Application” project with File → New → Project.
- Visual Studio will automatically generate some code, and present a screen that looks like this:



A step-by-step guide to building a simple GUI (cont.)

- The window labeled Form1 is the main window of the new application. Try running this program with F5 or Debug → Start Debugging. It's a boring program, but it runs and draws a window.
- Note that Form1 is being created by a class called `Form1`. We will return to this point.
- Open the Toolbox using View → Toolbox. (You can stop this pane from shutting using the pushpin icon.)
- Look in the Toolbox under "Common Controls," and drag a Button and a TextBox into Form1.
- Try running this slightly less boring, but still boring program.

A step-by-step guide to building a simple GUI (cont.)

- Use the Properties pane (View → Properties) to change the name of the TextBox to “myMessageBox”. Just as `Form1` is a class, the text box is field of `Form1` named `myMessageBox`.
- Similarly, change the button’s name to myButton and its Text property to “Push Me.” Note that changing the button’s Text property should result in an immediate change in the button’s appearance.

A step-by-step guide to building a simple GUI (cont.)

- Double click on the button. Visual studio will jump to the following code:

```
public partial class Form1 : Form
{
  public Form1()
  { InitializeComponent(); }

  private void myButton_Click(object sender,
                              EventArgs e)
  { }
}
```

- This is part of the `Form1` class definition. And `myButton_Click` is a method that will be called when the button is clicked.

A step-by-step guide to building a simple GUI (cont.)

- Add some interesting code to `myButton_Click`. For instance,

```
private void myButton_Click(object sender,
                            EventArgs e)
{
  this.myMessageBox.Text = "Go␣Quakers!";
}
```

- Run the program again, and make sure clicking the button changes the message.

Epilogue

Questions to ponder:

- To what extent was creating the form magic? To what extent can this be done purely in code?
- Assigning to `myMessageBox.Text` caused an effect to occur. How can an assignment do that?
- How does `myButton_Click` get run?