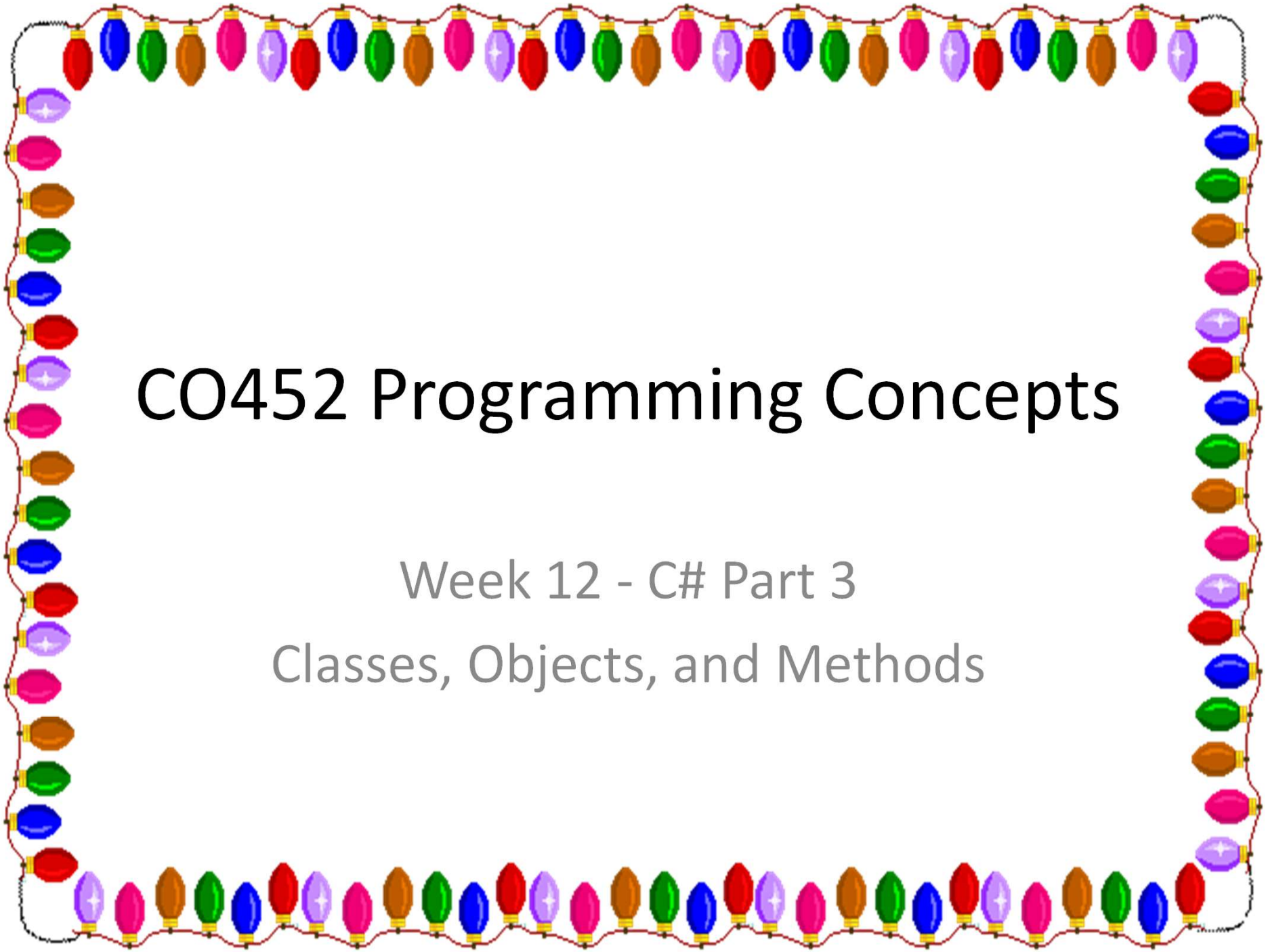


CO452 Programming Concepts

Week 12 - C# Part 3

Classes, Objects, and Methods

Object Oriented Programming (OOP)

What is Object-Oriented Programming?

- Uses Objects
- OOP programs model 'things' in the real world (objects)

  e.g.

  – a **banking** program has to model:

  customers, accounts, transactions (cheque, credit card)

  – a **medical** program has to model:

  patients, beds, ventilators, drugs, ...

  – sometimes we are modelling physical things in the real world

  – at other times the 'thing' doesn't exist outside the computer .. a more abstract concept

Classes and Objects

A **class** is like a plan or template for real-world **objects**

House Class

Class variables (attributes)

- address
- houseType
- age
- numberOfRooms
- owner

etc.

House Class

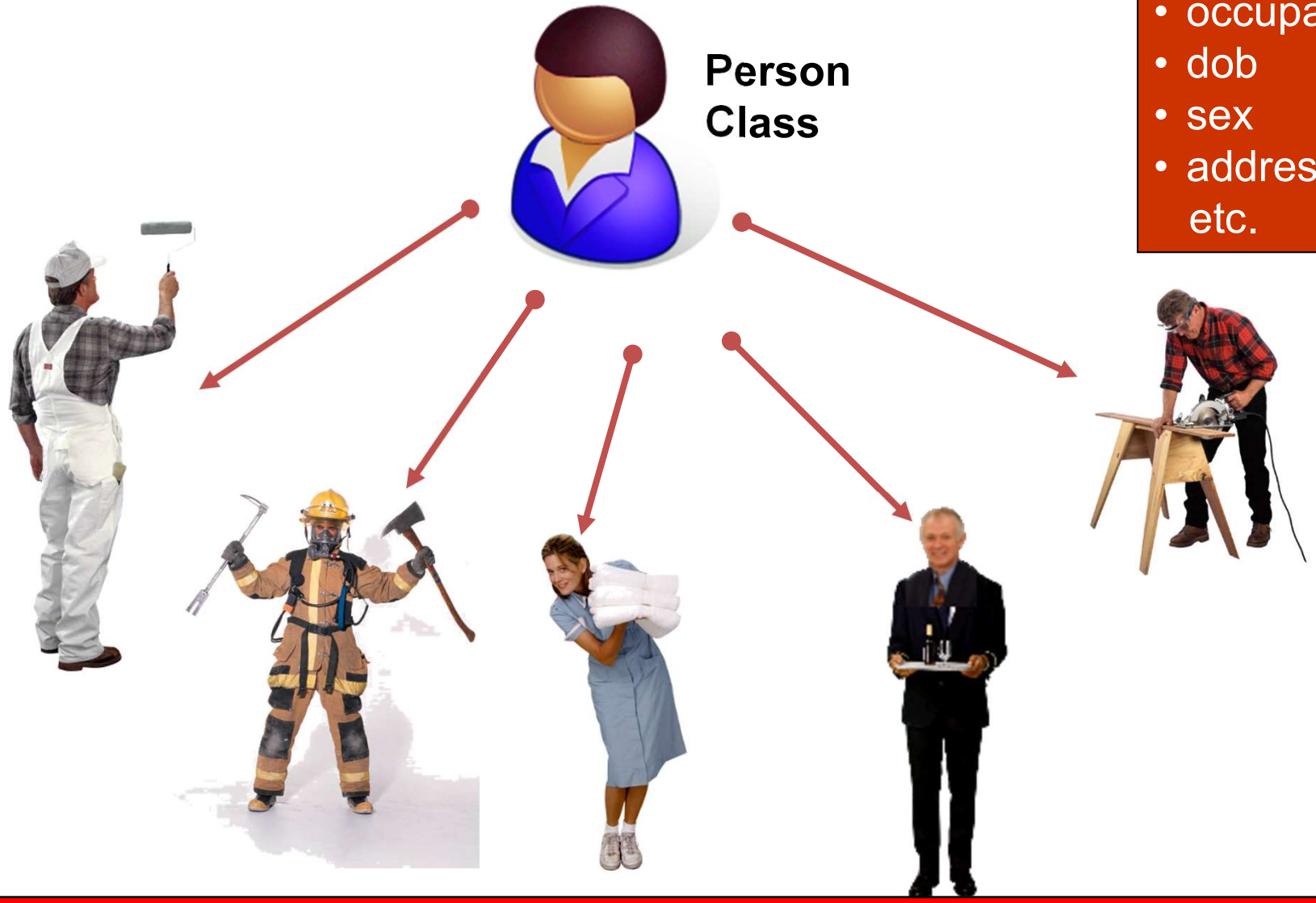
Any number of objects can be produced from a class

Person Class

Class variables (attributes)

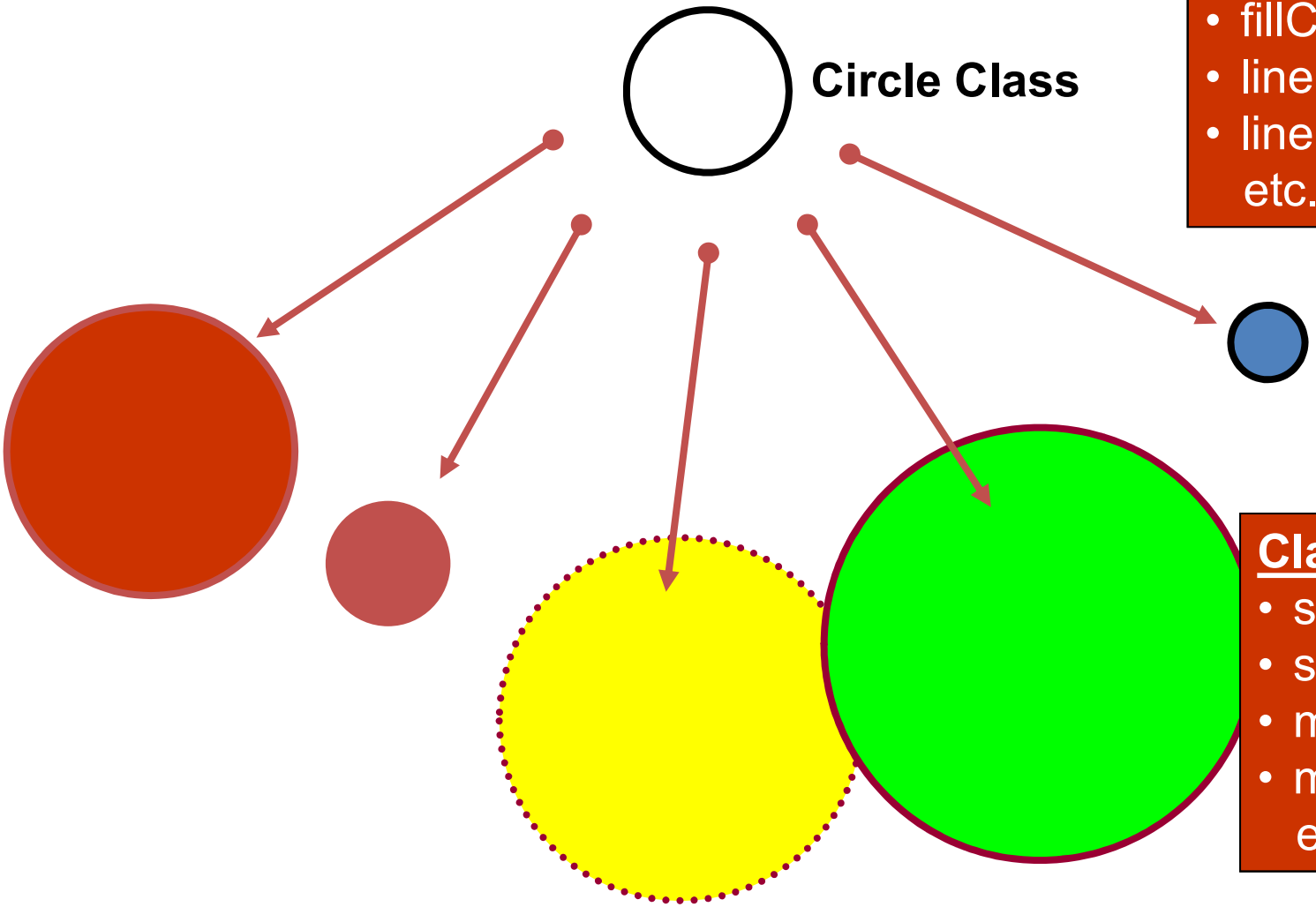
- name
- occupation
- dob
- sex
- address

etc.



Objects are also called **instances** of the class

Circle Class



Class variables
(attributes)
- radius
- fillColour
- lineColour
- lineStyle

etc.

Class methods
- setFillColour()
- setRadius()
- moveRight()
- moveUp()

etc.

Objects also have **methods (functions)** that define their behaviour: methods can do things with the object

Class

- sets of objects share attributes and behaviours
- all students have an id, a name, an address ... and attend lectures ☺
- it is more economical to define a class than to define everything independently for every object of that class (generalisation)

Class

- a class is like a type
- think of a class as a template or blueprint
  - A designer makes drawings for a **car** (class)
  - A factory can then manufacture individual cars (objects)
    - » **the blue Ford Focus 1.6L reg. no: YO54 7GD**
    - » **the red Ford Focus 2.0D reg. no: SH54 6TR**

Object

- an object models (represents) well defined entities (things)
- an object is an instance of a class
  - different objects are given different names (here derived from the ***student*** **class**)
  - **student1**, **student2**, **student3**, etc.

Object

- an object has attributes
  - (c# fields or instance variables)
- an object has behaviours (c# methods)

- attributes of the object, **student1** are:
  - **id**: 12345678
  - **name**: Fred Bloggs
  - **address**: 10 Happy Place, High Wycombe
  - **sex**: male
- behaviours of the object, **student1** are:
  - **attend lecture**
  - **sit exam**

Programming with classes and objects

C# is an OOP language

```
using System;
namespace  Task
{
    class MealCosts
    {
        static void Main ()
        {


        }
    }

}
```

Every C# project must have at least one class

and one of these classes must have a Main() method

Creating a new **Object** from the MealCosts Class

OR

```
MealCosts myMeals = new MealCosts();
```

Executing the **methods** of an object

First Create a new Object

```
MealCosts myMeals = new MealCosts();
```

Then you can call any of its methods

```
myMeals.inputData();
myMeals.calcTotalCosts();
myMeals.outputCosts();
```

Use the DOT operator

Example of Objects

Class →Three Objects (Instances)

TargetBot

List of variables:
Var1
Var2
Var3

object1

object2

object3

Objects are **unique instances** of a class structure

. Notation

. is otherwise known as the **'period caller'**

Refers to **variables** and **functions** from objects

objectName.Var1 **objectName.Function1**

. Notation

. is otherwise known as the **'period caller'**

Refers to **variables** and **functions** from objects

objectName.Var1 **objectName.Function1**

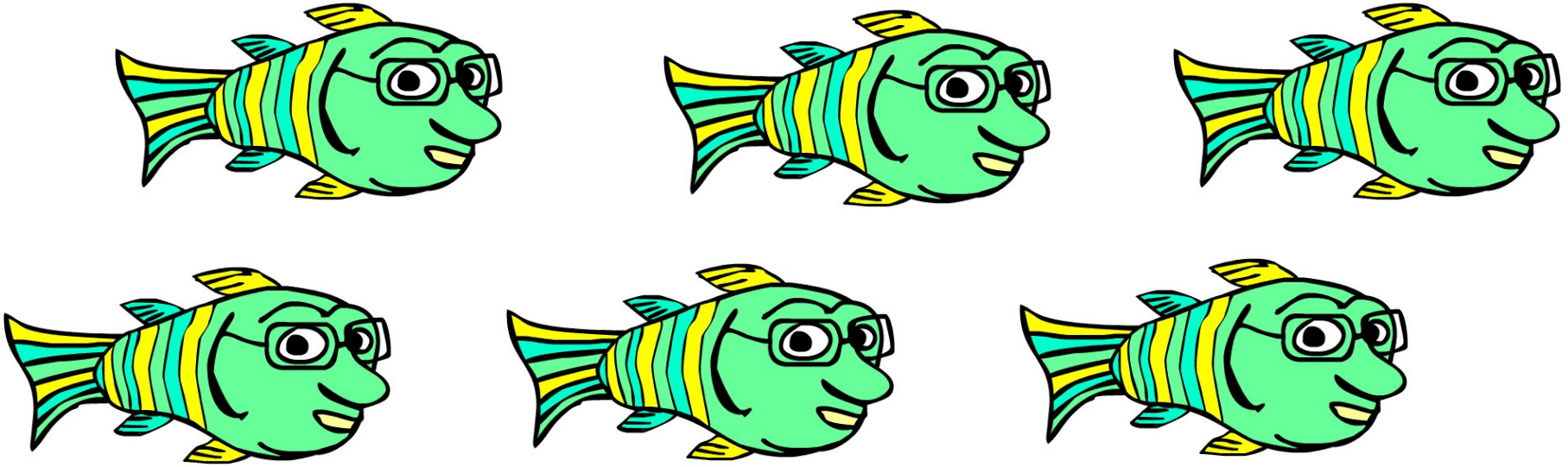
Examples from Ceebot:

item.category

this.position

In a Game

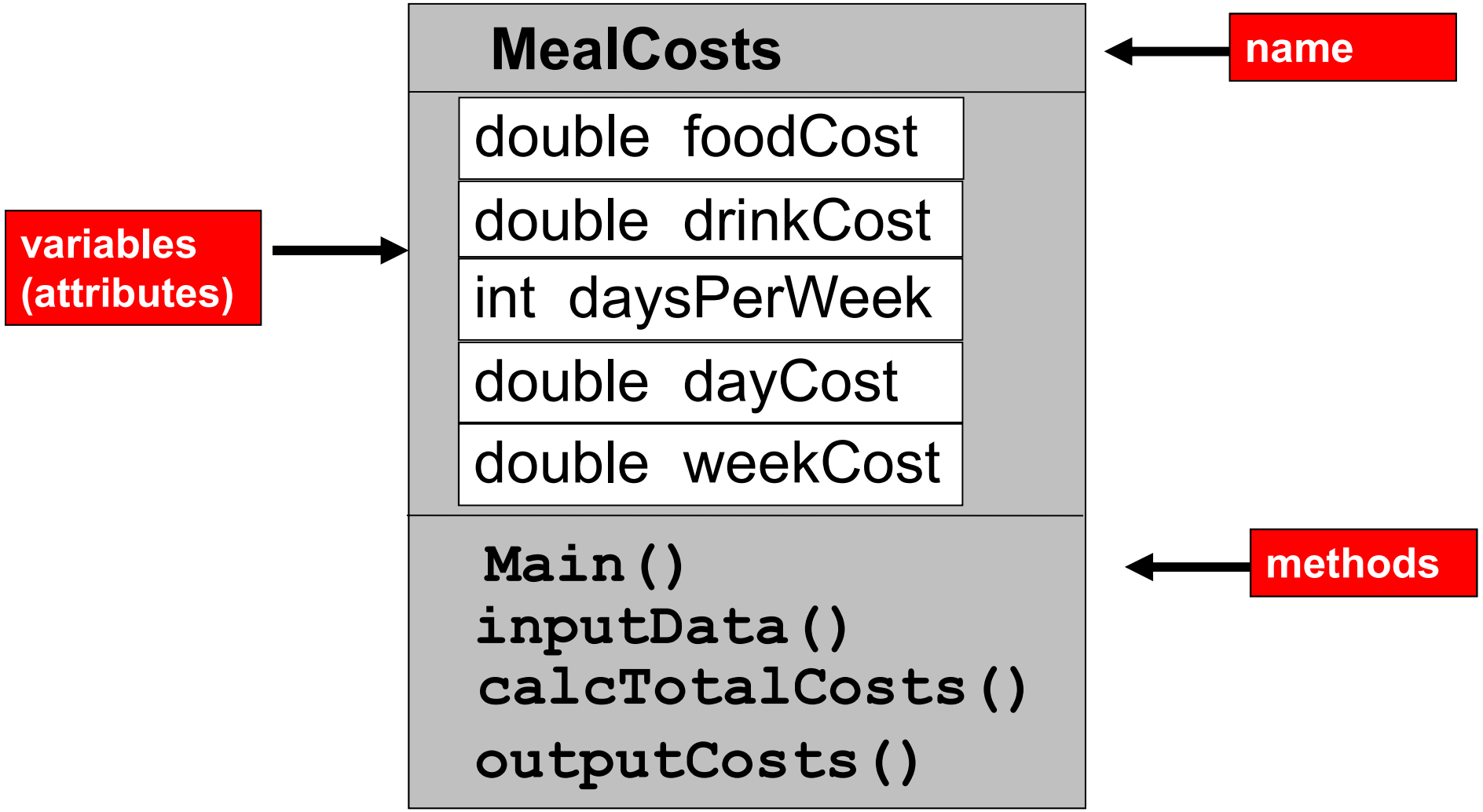
Several ***Fish objects*** could be generated or "spawned" from a ***Fish class***, with relevant ***variables and functions***



Programming a MealCosts Class

UML Class Diagram for a MealCosts Class

This shows all the data(class variables) and methods



MealCosts Class

```
class MealCosts
{
```

```
double foodCost, drinkCost;
double dayCost, weekCost;
int daysPerWeek;
```

Class variables (attributes)

```
static void Main()
{
    MealCosts myMeals = new MealCosts();
    myMeals.inputData();
    myMeals.calcTotalCosts();
    myMeals.outputCosts();
}
```

Main() Method

```
void inputData()
{
    string input;     // local input variable
    Console.Write("Enter the price of a meal: £");
    input = Console.ReadLine();
    foodCost = Convert.ToDouble(input);
    Console.Write("Enter the price of a drink: £");
      // etc.
```

myMeals inputData() method is called

MealCosts Class Methods (continued)

```
void calcTotalCosts()
{
    dayCost = foodCost + (3 * drinkCost);
    weekCost = dayCost * daysPerWeek;
}
```

```
void outputCosts()
{
   Console.WriteLine("\nYour Final Costing
   Results");
   Console.WriteLine("============================");
   Console.WriteLine("Total cost for one day = £"
                                    + dayCost );
   Console.WriteLine("Total cost for one week = £"
}                                   + weekCost );
```

```
}    //  end of the MealCosts class
```

Function Called from the Main() Program

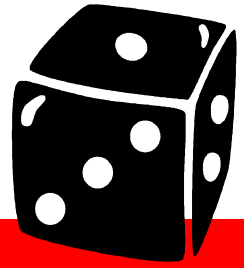
```
static void Main()
{
    function1();
}

static void function1()
{
    Console.Write("In function1");
}
```

A

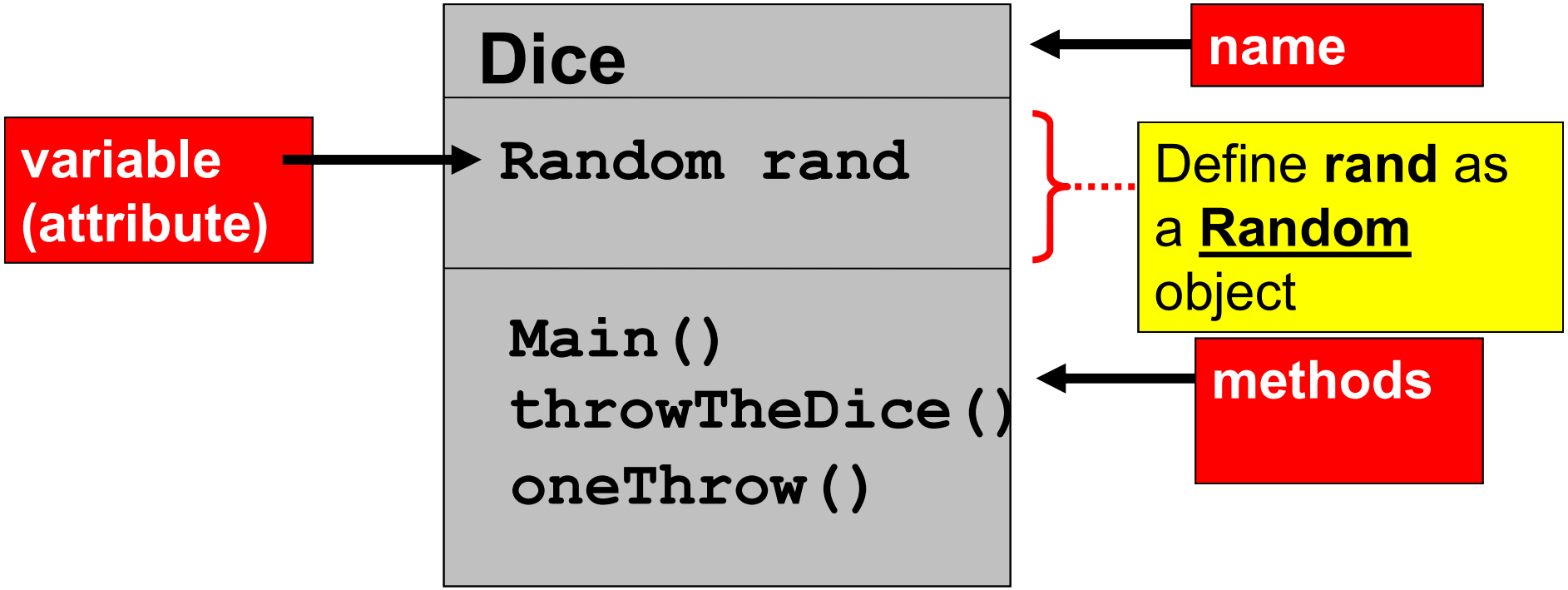
Dice

Simulator

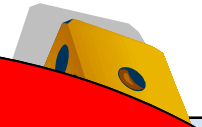
Dice Class Diagram (UML)



The class contains 1 variable and 3 methods



Definition of Dice class



```
class Dice
{
```

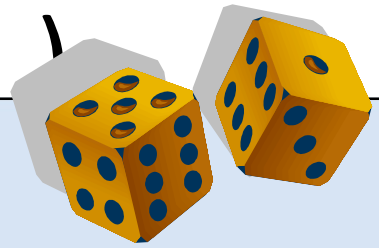
```
    private Random rand;
```

```
    public static void Main()
    {
        Dice myDice = new Dice();
        myDice.rand = new Random();
        myDice.throwTheDice();

    }
```

Note:
we can control access by making some things **private** and others **public**

other methods

Dice Class Definition – (contd)



```
public void throwTheDice()
{
   Console.WriteLine("I have thrown "
                              + oneThrow());
}
```

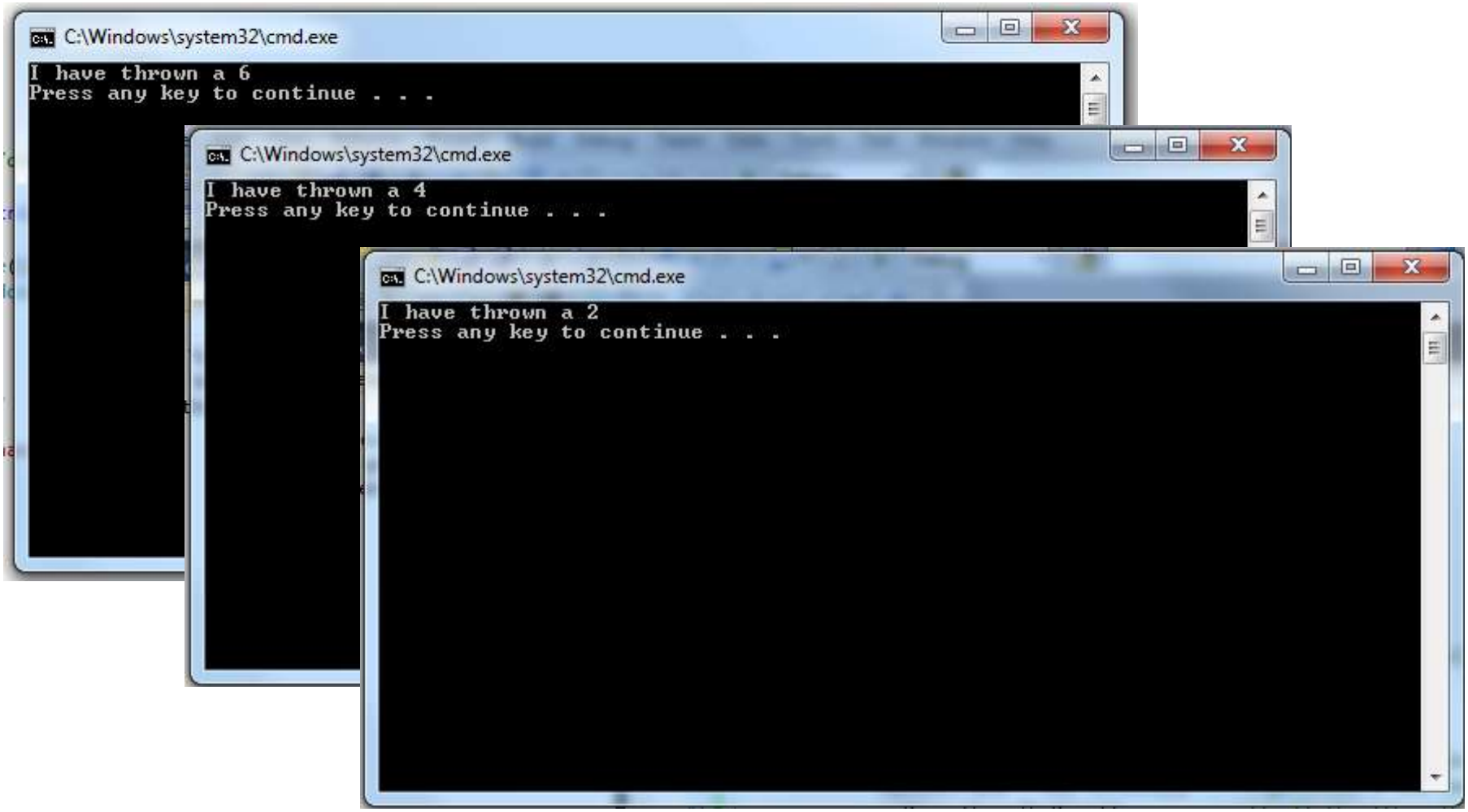
```
public int  oneThrow()
{
   return rand.Next(6)+1 ;
}
```

returns a random number (1-6)

```
}    // end of Dice class
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace DiceSimulator
7  {
8      class Dice
9      {
10         private Random rand;    //declare a Random variable called 'rand'
11
12         public static void Main(string[] args)
13         {
14             Dice myDice = new Dice();       //create a new object from the 'Dice' class
15             myDice.rand = new Random();     //generate a new Random object, associated with the variable called 'rand'
16             myDice.throwTheDice();          //call the 'throwTheDice()' method
17         }
18
19         public void throwTheDice()
20         {
21             Console.WriteLine("I have thrown a " + oneThrow());     //Output message including value returned...
22                                                                     //...from 'oneThrow()' method
23         }
24
25         public int oneThrow()
26         {
27             return rand.Next(6)+1;      //use the Random object to generate a number between 0 and 5 (Next(6))
28                                         //add ONE to the number generated so that it is between 1 and 6
29                                         //RETURN value where it is needed in the 'throwThe Dice()' method
30         }
31
32     }
33 }
34
```

Example Outputs



Private and Public

private

This word is used to prevent access from outside the class

- normally class variables are defined as private

  e.g. **private string choice;**

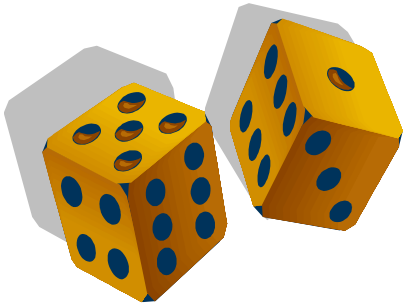
public

This word is used to make a method available to the world outside the class

e.g. **public void throwTheDice()**

Multiple Dice Throws

Throw Dice 6 Times?



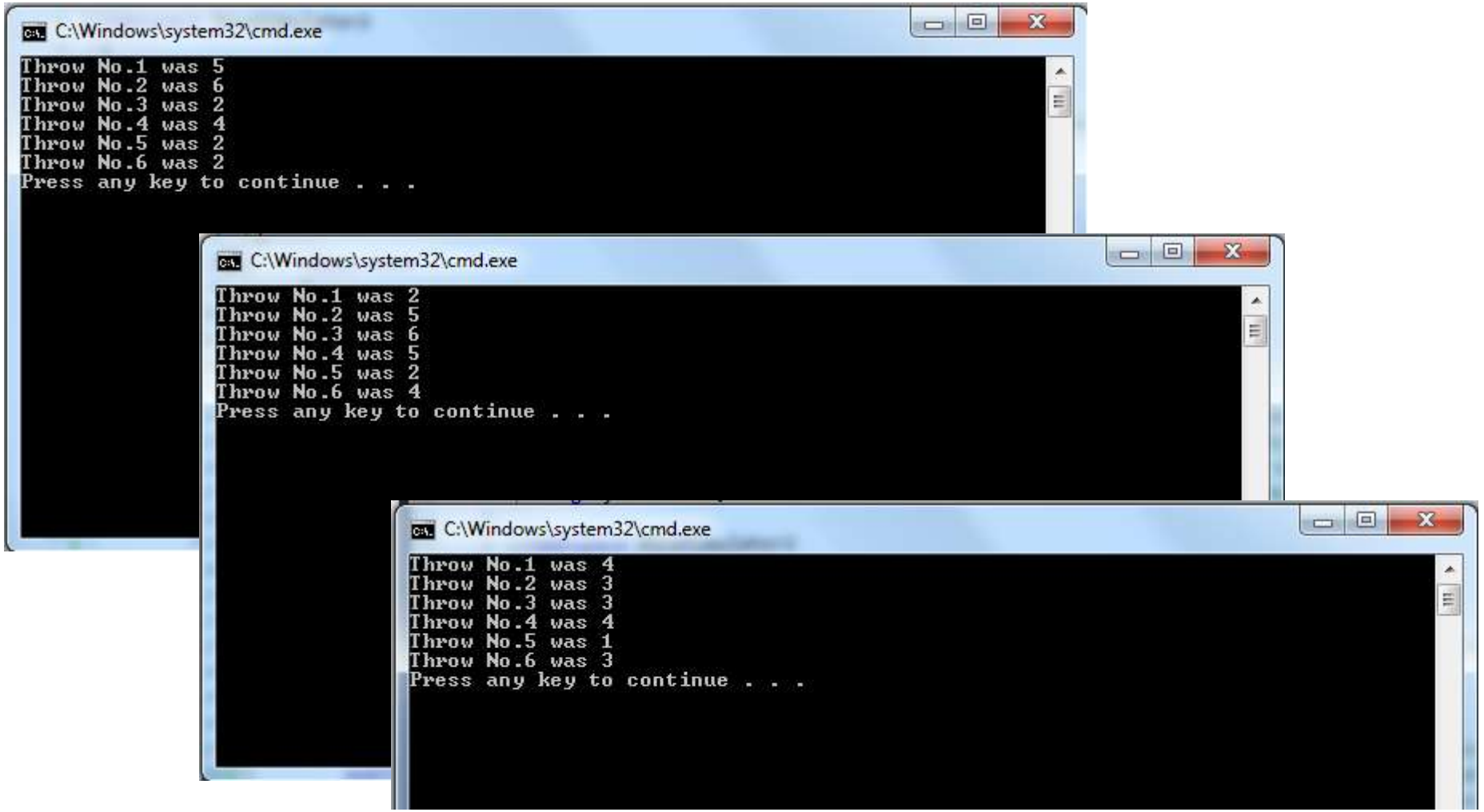
// New version of throwTheDice() method

```
public void throwTheDice6Times()
{
   for (int i = 0; i < 6; i++)
   {
      Console.WriteLine("I have thrown "
                        + oneThrow());

   }
}
```

Call the **oneThrow()** Method (6 times)

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace DiceSimulator2
7  {
8      class Dice
9      {
10         private Random rand;        //a 'Random' type variable called 'rand' is declared
11
12         public static void Main(string[] args)
13         {
14             Dice myDice = new Dice();       //a new object is generated from the 'Dice' class....
15                                             //....associated with the variable called 'myDice'
16             myDice.rand = new Random();     //a new Random object is generated, associated with the variable called 'rand'
17             myDice.throwTheDice6Times();    //call the 'throwTheDice6Times()' method from below
18         }
19
20         public void throwTheDice6Times()
21         {
22             for(int i=0; i<6; i++)          //set up a FOR Loop to repeat 6 times
23             {
24                 Console.WriteLine("Throw No." + (i + 1) + " was " + oneThrow());
25             }
26         }
27
28         public int oneThrow()
29         {
30             return rand.Next(6) + 1;    //this method is called 6 times, each time RETURNing a new random value
31         }
32     }
33 }
34
```

Example Outputs



The Constructor

- The constructor is a special method in a class
- It always has the same name as the class
- When an object is created from a class, the constructor is automatically executed
- It is used to initialise the new object

```
public Dice()
{
   rand = new Random();
}
```

*the **new** operator*

***constructor** method*

This constructor creates a new Random object, used to generate random numbers for the Dice

return

- **return** is used to return a value back from a method or function
- using **void** in the method header means that nothing is returned by this method
- If return is used you must change void to show the type being returned (e.g. int, string, double, etc.)

```
public  int  oneThrow()
{
  Return rand.Next(6)+1;
}
```

oneThrow() returns a random integer value .. so change **void** to **int**

Quiz

What is the difference between a class and an object?

The Last Slide

Extra Reading

An example using 2 Classes

- **NuclearStation**: a class to control a Nuclear Power Station
- **Test**: a class to test this

a Test class (incomplete)

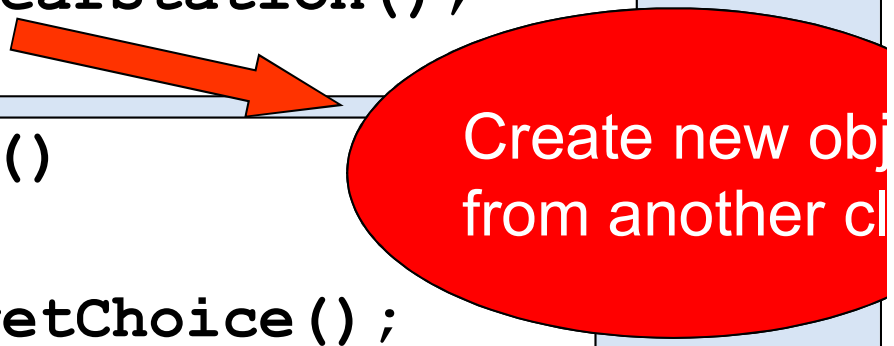
```
class Test
{
```

```
private string choice;
private NuclearStation myStation;
```

```
public static void Main()
{
    Test myTest = new Test();
    myTest.testStation();
}
```

```
public Test()  // the class constructor
{
    myStation = new NuclearStation();
}
```

```
public void testStation()
{
    myStation.display();
    choice = myStation.getChoice();
    if (choice == "1")
        myStation.lowerRods();  // etc.
```



NuclearStation class (inco

Note: this class has no Main() method!

```
class NuclearStation
{
    private const string SYSTEMCODE = "NUKEME";

    public void display()
    {
        Console.WriteLine("Nuclear Winter Station");
        Console.WriteLine("    Main Menu ");
        Console.WriteLine(" 1: Lower Fuel Rods ");
        Console.WriteLine(" 2: Raise Fuel Rods ");
        etc.
    }
    public string  getChoice()
    {
        string choice;
        Console.WriteLine("What do you want to do?");
        Console.Write("Enter your choice:");
        choice = Console.ReadLine();
        return choice;
    }
```

The NuclearStation class (continued)

```
public void lowerRods()
{
   string code;
   Console.WriteLine("Danger:Lowering Fuel
   Rods");
   Console.Write("Enter authorisation code: ");
   code = Console.ReadLine();
   if (code == SYSTEMCODE)
      Console.WriteLine("CODE CORRECT\nRods now
   being lowered");
   else
      Console.WriteLine("CODE INCORRECT:\nYou
   will now be escorted  from the building");

}
```

```
} // end of NuclearStation
class
```