

# Coding for Game Development

## Section Five

### XNA Game Programming

#### Object Orientation in C# [Part Two]

1 / 21

# Visibility

- Every item in a class (fields, methods, constructors, even classes themselves) can optionally be marked with a *visibility*, indicating whether or not it can be seen by the "outside" world.
- For the XNA programmer, we really only need to consider three visibilities:
  - **public**
  - **private**
  - **protected**
- If we omit a visibility then C# usually assumes **private**.

2 / 21

# Access Modifiers

- Access modifiers in C# have the following meanings:
  - **public** The item can be accessed globally.
  - **private** The item can only be accessed within the class in which it is defined.
  - **protected** The item can be accessed within the class in which it is defined and any child classes.
- If omitted, the usual default is **private**
- Unlike other languages, it is common practice in C# to omit the visibility modifier for fields and methods so they default to private.

3 / 21

# Void and Non-Void Methods

- Methods can be either *void*, or they can return a value:

## Method Definition

```
public void VoidMethod()  
{  
    ... some statements ...  
}
```

## Method Call

```
bobsCar.VoidMethod();
```

## Method returns int value

```
public int IntMethod()  
{  
    int theAnswer = 12;  
    return theAnswer;  
}
```

```
int v = bobsCar.IntMethod();
```

Variable to receive the returned value

4 / 21

# Variables

- A variable is similar to a field, but with two important differences:
  - A variable is defined *inside* a method (A field is defined inside a class)
  - A variable only exists *when the method is running* (A field persists while the object exists)
- It is extremely bad programming practice to use a field when a variable would suffice! It is also a very common student mistake!

5 / 21

# Method Parameters

- Parameters are a way of passing values into a method when it's called.
- Parameters must be defined in the method heading.
- A value for each parameter must be specified at the point where the method is called.

```
int _gear;  
public void UpGearBy(int amount)  
{  
    _gear = _gear + amount;  
}
```

```
bobsCar.UpGearBy(2);
```

- A parameter is a variable which is pre-initialised to the specified value at the start of the method.

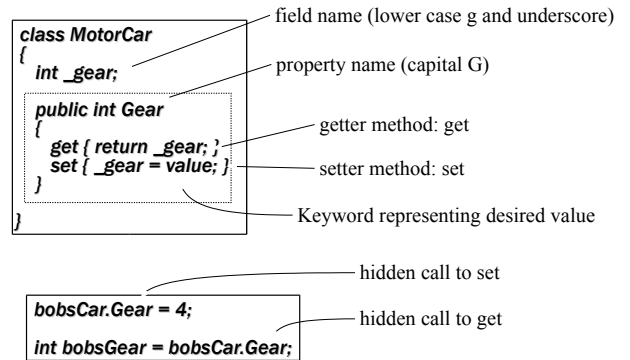
6 / 21

## Properties (Smart Fields)

- Good programming practice dictates that fields should never be declared *public*.
- Instead they should be declared **private** (or **protected**) and only accessed by other classes by using special *getter* and *setter* methods.
- C# has a unique feature which makes this process very simple to set up.
- For each field we can create a public *property* which is used to access the field.
- The property usually has the same name as the field, starting with a capital letter.

7/21

## Creating and Using Properties



8/21

## Static Fields and Methods

- Fields and methods can optionally be marked **static**.
- Unlike normal fields, a static field uses the same memory location for every object of the class.
- This means every object sees the same value for the static field; if one object changes the value, then it is changed for all objects.
- A **static** method can be called before any objects of the class have been created.
- Static fields and methods are not greatly used in XNA, but they are an important concept in more general OO programming.

9/21

## Objects and Pointers

- Creating an object is a two step process:
  - Declare an object variable (or field)
 

```
MotorCar bobsCar;
```
  - Construct the object
 

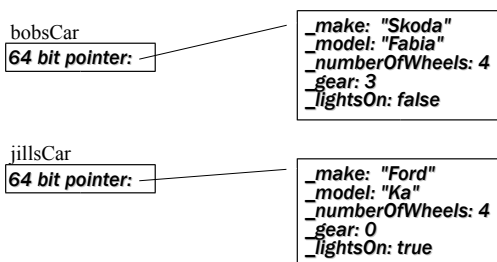
```
bobsCar = new MotorCar(...);
```
  - Frequently, we combine both steps into one statement:
 

```
MotorCar jillsCar = new MotorCar(...);
```
- The declared object variable is a 64 bit value.
- When we create an object, memory to store its fields is taken from a part of memory called *The Heap*.
- A pointer to the object is stored in the variable.

10/21

## Pointers to Objects

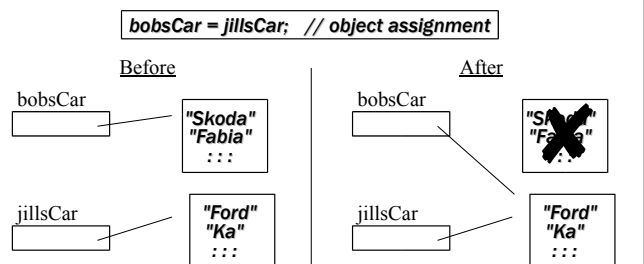
- The declared object variables (`bobsCar`, `jillsCar`) are 64 bit values which store pointers to each object.



11/21

## Assignments of Objects

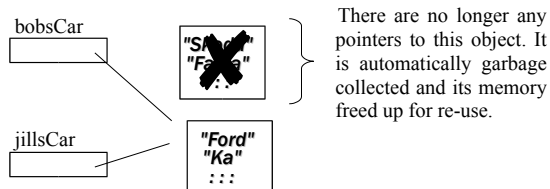
- (Assignment is the setting of a value to a variable/field using the = operator)
- When assigning one object to another, we actually copy the pointer value:



12/21

## Automatic Garbage Collection

- When there is no pointer to an object, that object is destroyed. The programmer is not required to explicitly release the object's memory back to the system.
- Unlike C++, destruction and garbage collection is completely automatic. This makes C# programming considerably easier than C++



13/21

## Null Pointer

- We can destroy an object by assigning null to its (only) object variable:

```
bobsCar = null; // will invoke garbage collection
```

- An object variable contains null when first declared:

```
MotorCar nullCar; // contains the value null
```

- We can test an object variable to see if it is null:

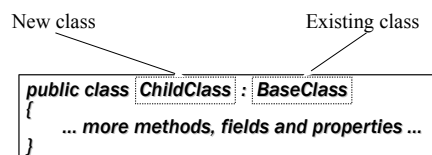
```
if (bobsCar == null) { // test if pointer is null
    // The object doesn't exist
}
```

- It is an exception to try to access an object using a pointer which is null: Null Pointer Exception.

14/21

## Inheritance

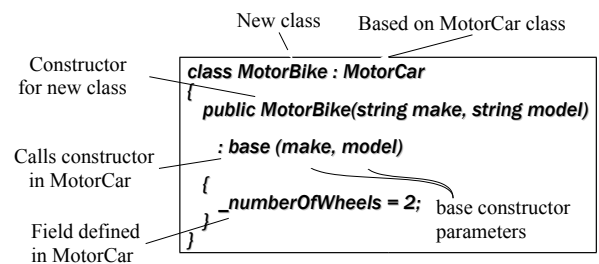
- Inheritance is an important part of C# class design:
  - Inheritance allows us to design a new class based on an existing (or *base*) class.
  - The new class retains all the behaviour of the base class, but we can add new fields and methods, or redefine existing methods.



15/21

## Constructors in Child Classes

- If the base class has a constructor, then child class must also have a constructor.
- The child class constructor must use the word **base** to access the base class constructor:



16/21

## Protected Visibility

- The constructor for the new class won't compile!
- The visibility of **numberOfWheels** in the base class is set to the default of private.
- The child class can't access it because of this private visibility.
- We must change the visibility of **numberOfWheels** from private to **protected**.
- This change must be made in the base class!
- Protected visibility means that the item can be seen by child classes, but it is still not publicly visible to other classes.

17/21

## Using a Child Class

- Once we have a child class, we can use it like any other:

```
MotorBike bobsBike = new MotorBike ("Yamaha", "650");
```

- The methods defined in the base **MotorCar** class are also available to the child **MotorBike** class:

```
bobsBike.GearUp();
bobsBike.GearDown();
bobsBike.Gear = 4;
```

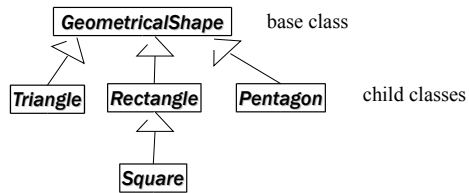
} GearUp(), GearDown() and Gear are all defined in the base MotorCar class

- Any new methods added to the child class are (of course) available for use by any of the child objects.

18/21

## Redefining a Method

- An important part of OO design is the ability for a child class to redefine a method in the base class.
- Consider a class hierarchy of geometrical shapes:



- Each class can define its own **Area()** method!

19/21

## Overriding a Method

- To redefine a method the following must happen:
  - The *base* class method must be marked **virtual**
  - AND*
  - The child class method must be marked **override**.
  - The child class method must have the same parameter list as the base class method.
- We will see examples of this in later lectures when we develop an Object Oriented class hierarchy for sprites.

20/21

21/21