SCC.111 Software Development – Lecture 29: OO Fundamentals in Java

Adrian Friday, Hansi Hettiarachchi and Nigel Davies

Introduction

Lancaster University

- Last lecture, we looked at:
  - The role of virtual machines in programming languages
  - A brief introduction to Java

- Today we're going to learn about
  - How to apply the core Oriented Object concepts in Java
  - The Java Class Library
  - How to automatically document our code using JavaDoc

Core OO concepts in Java

Lancaster University

- **Encapsulation principles and keywords remain the same as C++**
  - **Classes**
    - Use the class keyword to define class.
    - Create in a .java file matching the name of the class.
    - Almost always declare as public.
    - **There is no header file**
  - **Attributes**
    - Create as variables inside the class.
    - Almost always declare as private.
  - **Methods**
    - Create inside the class
    - May be declared public or private
    - **Implementation also goes inside the class definition**

Core OO concepts in Java…

Lancaster University

- **Encapsulation principles and keywords remain the same…**
  - **Constructors**
    - Create as a standard method inside the class
    - Method name must match the name of the class
    - No return type defined.
    - Multiple constructors are permitted, provided the parameters are different
  - **public**
    - Access modifier that states the following method/variable is accessible to all code.
    - Add the **public** keyword **explicitly** before **each** method/variable
  - **private**
    - Access modifier that states the following method/variable is accessible only to code inside the class.
    - Add the **private** keyword **explicitly** before **each** method/variable
  - **Composition**
    - Classes may hold attributes of class type, not just primitive types….

An Example Java Class…

Lancaster University

```
public class Car {
    // Instance variables go here
    private int milesDriven;
    private String colour;

    // Methods go here
    public void drive (int miles) {
        milesDriven = milesDriven + miles;
    }

    public Car (String newColour) {
        colour = newColour;
        milesDriven = 0;
    }
}
```

Creating Object Instances: **new**

Lancaster University

- Just like C++, once defined, classes are **types** and can be realized into **object instances**.
  - **By yourself, or by other programmers you give your class file to!**
  - A realized class is called an **object instance** and is treated just like any variable

- In Java object instances are built from a class by using the **new** keyword:

```
public static void main( String[ ] arguments ) {
    Car c = new Car("white");
}
```

Class name

The name of your new variable.
Which is actually an **object reference...**

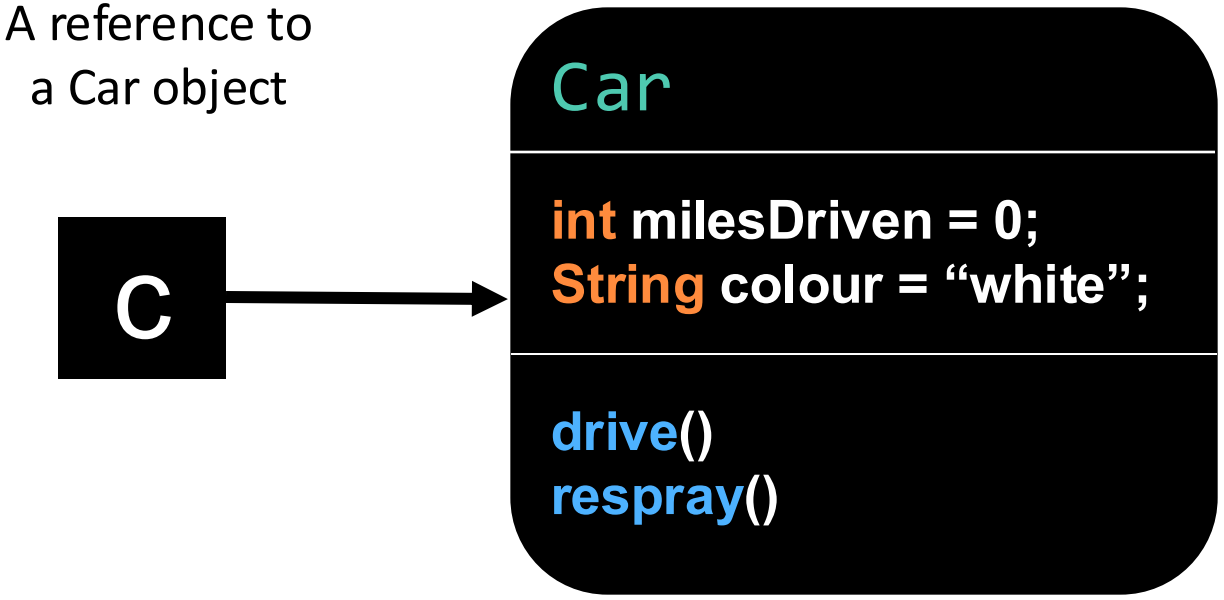
The constructor for the new object

Java Object Creation

Lancaster University

```
Car c = new Car("white");
```

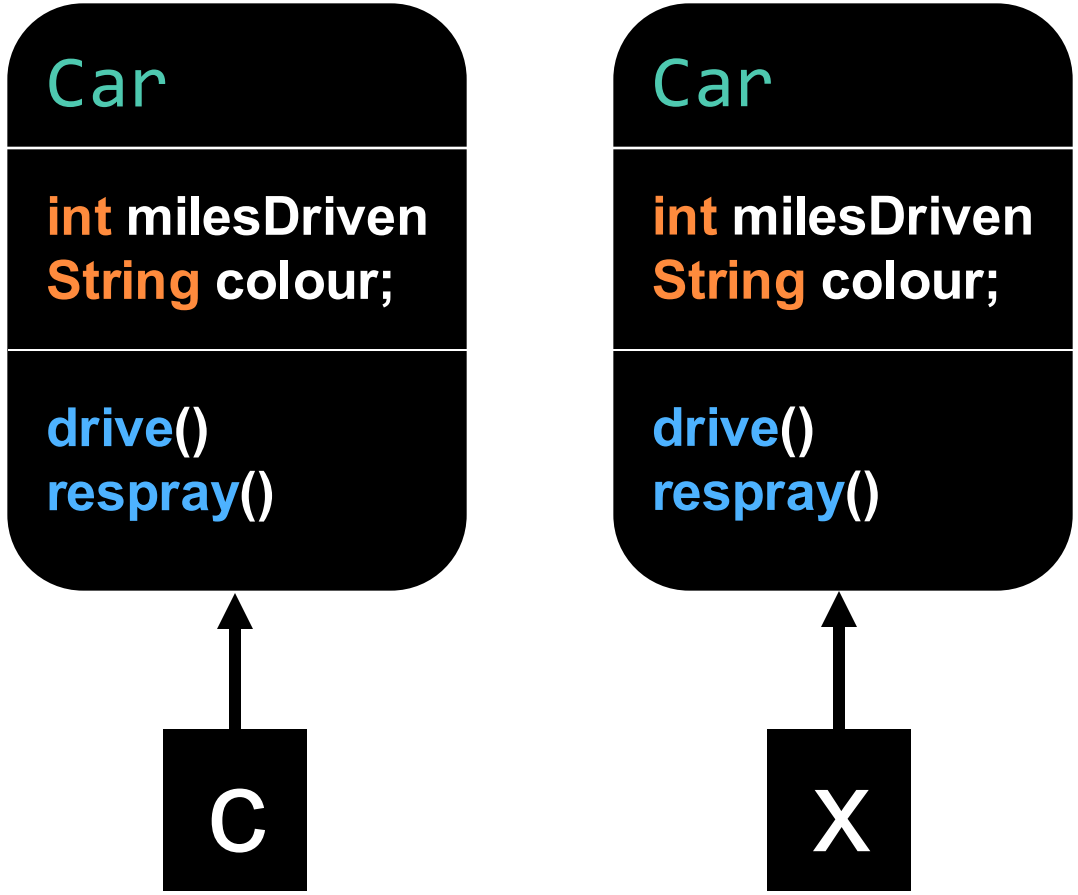
A reference to a Car object



Java Object Referencing

Lancaster University

```
Car c = new Car();
Car x = new Car();
```

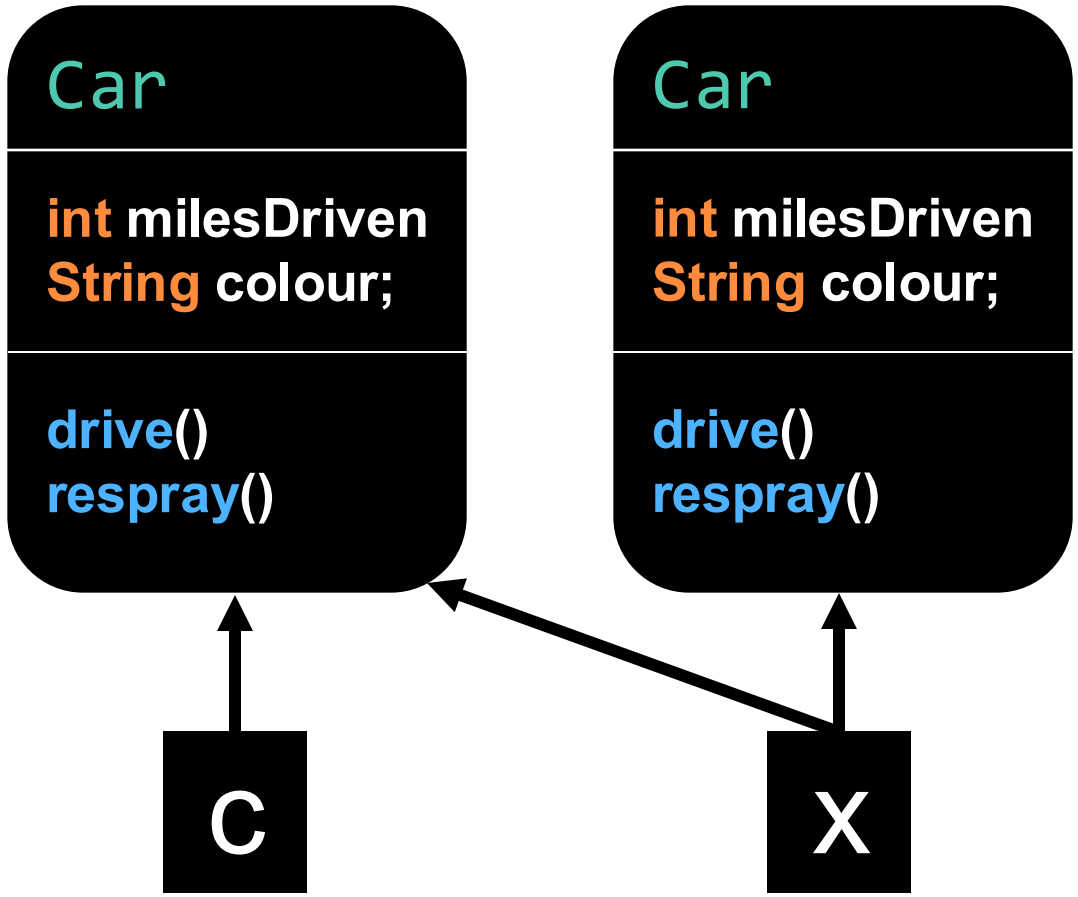


Java Object Referencing 2

Lancaster University

```
Car c = new Car();
Car x = new Car();

x = c;
```



Java objects are automatically garbage collected (deleted) when they have no references…

Value Types and Reference Types

Lancaster University

- **Java is a 100% pass by value language.**

- **Primitive types** are also known as **Value Types**. When used, it is the data itself you are accessing/modifying or passing into a function.
- Primitive types can be identified as they use pure lower case.
- E.g. int, char, float, double, boolean…

- **Everything else** is a **Reference Type**. These use object references to objects. Whenever variables of this type are used, it is the **reference** that you are accessing/modifying.
- **Reference Types** can be identified as they (should) use capitalized camel case.
- E.g. Car, String, FluffyElephant, Minion, CatDog…

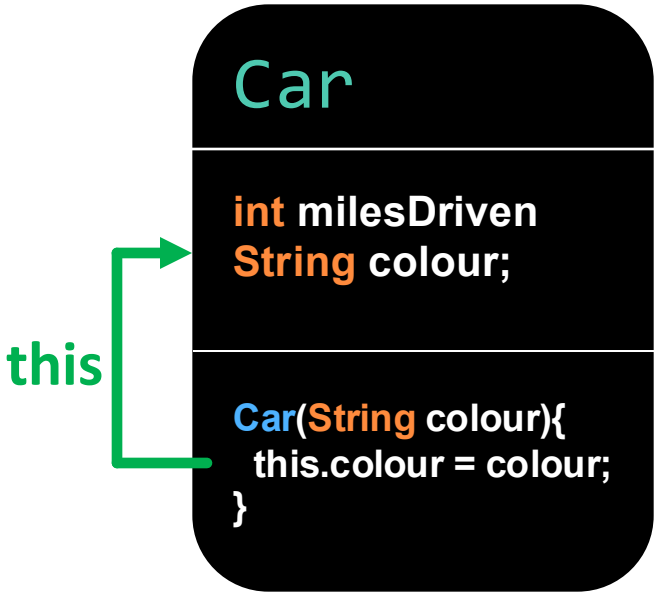
The **this** keyword in Java

Lancaster University

- Java has a special keyword that returns an object reference to the object instance which the current method is executing in - **this**
- **this** can be used to unambiguously refer to methods and variables:
- which can be particularly useful in constructors...
- ... but also anywhere we want to be explicit...
- ... or anywhere we want to pass the current object as a parameter.
- (C++ also has **this**, but it's a pointer!)

```
this.drive(4);

if (this.milesDriven>18)
…
```



The Java Class Library

Lancaster University

Java includes a vast set of pre-written classes known as a class library

- Examples you've already seen include: String, System, Math
- This is documented at: https://docs.oracle.com/en/java/javase/23/docs/api/
- This includes full documentation on the methods each class provides

Example: The String Class 1

Lancaster University

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods** | **Deprecated Methods**

Modifier and Type	Method and Description
char	**charAt**(int index) Returns the char value at the specified index.
int	**codePointAt**(int index) Returns the character (Unicode code point) at the specified index.
int	**codePointBefore**(int index) Returns the character (Unicode code point) before the specified index.
int	**codePointCount**(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
int	**compareTo**(**String** anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(**String** str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(**String** str) Concatenates the specified string to the end of this string.
boolean	**contains**(**CharSequence** s) Returns true if and only if this string contains the specified sequence of char values.

Example: The String Class 2

Lancaster University

```
public static void main( String[ ] arguments ) {
    String s = "Hello World";
}
```

```
public static void main( String[ ] arguments ) {
    String s = new String("Hello World");
}
```

Example: The String Class 2

Lancaster University

```
public static void main( String[ ] arguments ) {
    String s = "Hello World";

    // Examples of invoking methods on an instance

    int len = s.length();

    String l = s.toLowerCase();

    char[] myArray = s.toCharArray();
}
```

API Documentation

Lancaster University

Programmers document their classes, so other programmers can learn how to use them in their applications.

- Java uses a standard called **JavaDoc**
  - All Java programs use this, so they all have documentation in the same common format.
  - This makes it very easy to learn about other people classes which promotes reuse!

API Documentation: JavaDoc

Lancaster University

- Documentation is embedded inside your code using comment blocks
  - Remember Java can use /* */ comments like C
- JavaDoc uses **/** **/** to identify a JavaDoc comment
  - Additional '*' characters on following lines just make it look pretty
- JavaDoc is used to document all public classes, methods and constructors
- Comment block goes immediately before the item it is documenting

```
/**
* The Car class represents all drivable vehicles with four wheels.
* The class is also capable of modelling the outward appearance,
* car model and the number of miles driven.
**/
public class Car {
}
```

API Documentation: JavaDoc 2

Lancaster University

- Keywords allow structured documentation of methods, return values and required parameters
  - **@param <name>** documents the meaning of the parameter called ‘name’
  - **@return** documents the meaning of a value returned from a method
  - Many more: https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

```
/**
  * Creates a new Car with the given characteristics.
  * @param col The colour of the car being created.
  * @param miles The number of miles the car has driven.
 **/
public Car(String col, int miles) {
    colour = col;
    milesDriven = miles;
}
```

API Documentation: JavaDoc 3

Lancaster University

JavaDoc can then generate written documentation automatically!

- The semi-structured comments and the Java code are processed to build web pages documenting your classes
- These can then be posted online let other programmers know how to use your classes!

- Simply run the javadoc tool from the command line to generate the web pages
  - **javadoc –d doc *.java**
  - The "-d" parameter specifies the directory (folder) to store the web pages in
  - After running javadoc, just double click 'index.html' in this folder
  - Recommend always specifying a folder – Javadoc produces a lot of html!

Lancaster University

Live demo…

Practice Task

Lancaster University

- Try rewriting the Week 12 Dice exercise in Java…

Tip: Random number generation:

```
import java.util.Random;
public class Dice{
    public static void main(String[] args) {
        Random rand = new Random();

        // generate a number between 0 and 19
        int my_rand = rand.nextInt(20);
    }
}
```

Summary

Lancaster University

- Today we learned:
  - How the Oriented Object concepts we've already learned are applied in Java.
  - Java has no concept of header files and defines a classes methods and attributes together.
  - Java finds classes by searching by filename.
  - Any variable this isn't of a primitive type is an object.
  - Java uses mutable object references rather than pointers.
  - There are many classes already in the Java Class Library for you to use.
  - How to automatically create your own documentation in JavaDoc.