

GC01 Introductory Programming

Week 3



Dr Venus Shum
Department of Computer Science

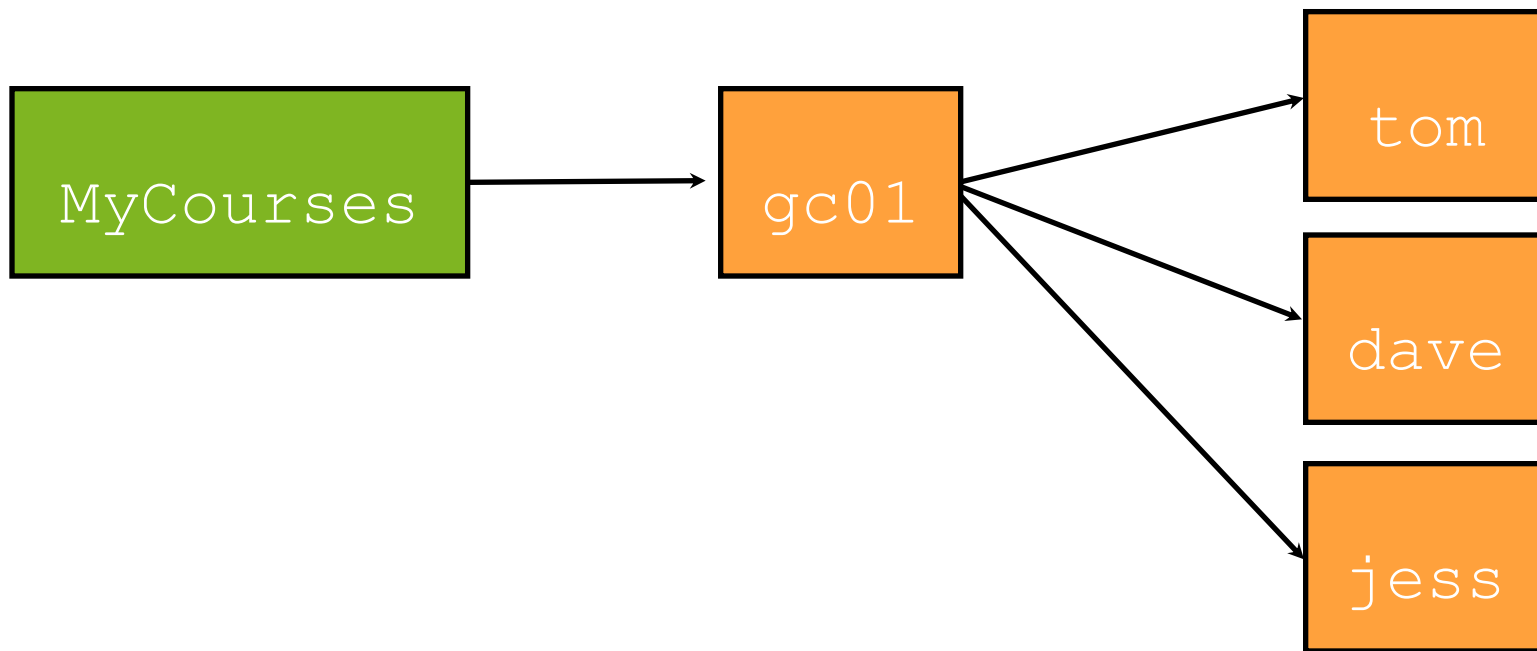
Pop quiz

- Methods and fields that apply to classes rather than objects are called _____ methods/fields
- Object is an _____ of a class
- Keywords “private/public/protected” are called _____.
- Classes that are primary for grouping related fields methods are called _____.
- The other 2 types of classes we have learnt are: _____
- The good practice is to declare fields in a class as _____ and use “getter” and “setter” methods to access and change them.

Review on Class and Objects

Module

Student



```

2
3 public class Main {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         int numStudents = 93;
8
9         Module myCourses = new Module("GC01", numStudents);
10
11         // create an object for every students in GC01
12         for (int i=0; i<numStudents; i++) {
13             myCourses.myStudents[i] = new Student(i);
14         }
15
16         //Use individual object to store information - manipulating objects
17         myCourses.myStudents[0] = new Student("Amy");
18         myCourses.myStudents[0].createLabWork(6);
19         myCourses.myStudents[0].getLabwork(0).setResult(90);
20         myCourses.myStudents[0].getLabwork(1).setResult(80);
21         myCourses.myStudents[0].getLabwork(2).setResult(85);
22         myCourses.myStudents[0].getLabwork(3).setResult(75);
23         myCourses.myStudents[0].getLabwork(4).setResult(80);
24         myCourses.myStudents[0].getLabwork(5).setResult(80);
25
26         System.out.println("Final result of "
27             + myCourses.myStudents[0].getName() + " is: "
28             + myCourses.myStudents[0].getFinalScore());
29
30     }
31
32 }
33

```

Create 1 "Module" object

Create 93 "Student" objects

Topics still to study

- ArrayLists
- Strings
- Inheritance
- Math functions
- File Handling
- GUIs and Graphics
- Writing our own Data Structures

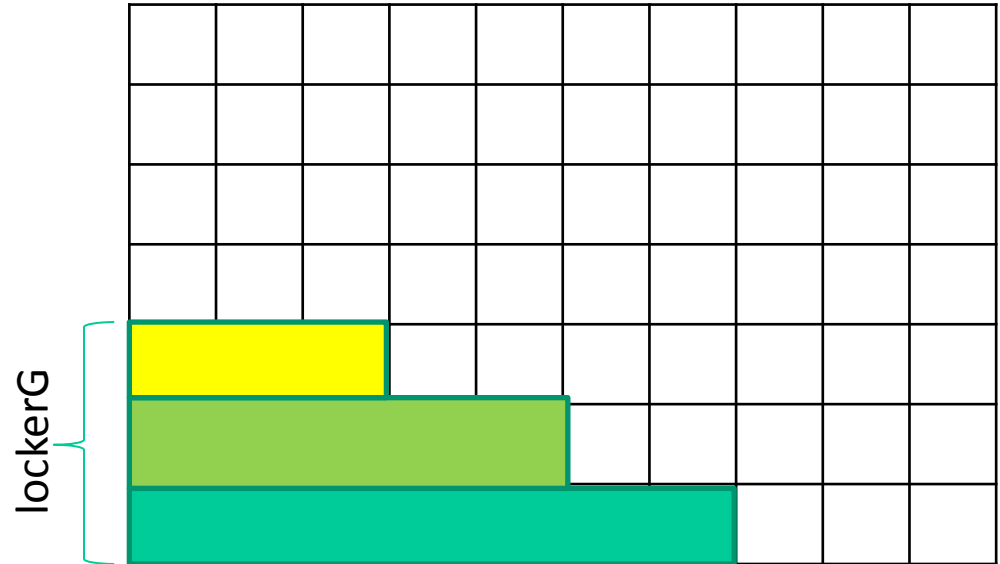
ARRAY LIST

Remember this array example:

Find lockers for students – laptop

- 1st year students - 7
- 2nd year students - 5
- 3rd year students - 3

In real world	JAVA
What are these lockers for? (type&size)	<code>String[][] lockerG;</code>
Assign 3 rows of them	<code>lockerG = new String[3][];</code>
Assign one row for 1 st year	<code>lockerG[0] = new String[7];</code>
Assign one row for 2 nd year	<code>lockerG[1] = new String[5];</code>
Assign one row for 3 rd year	<code>lockerG[2] = new String[3];</code>



This is index

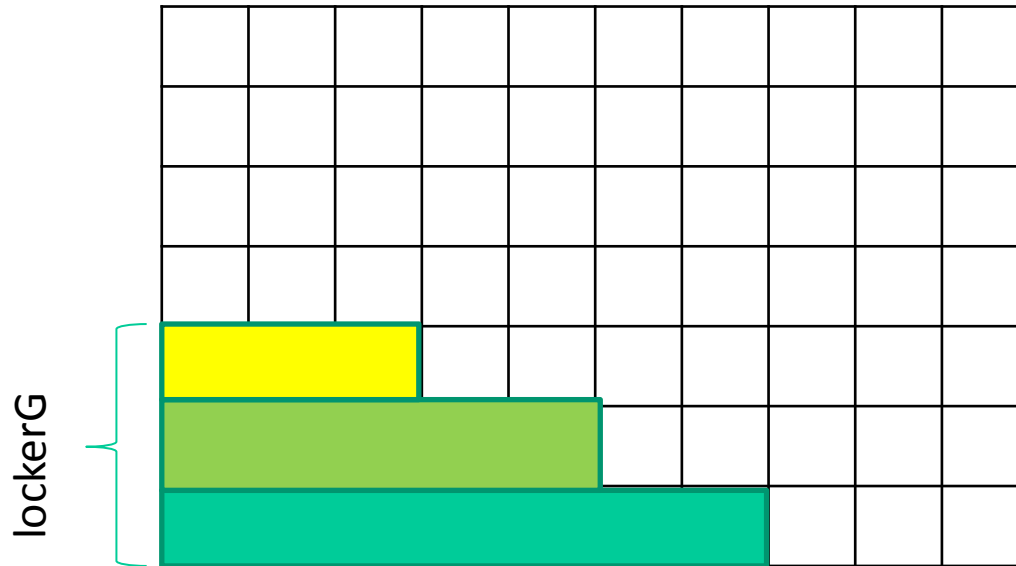
See how the index starts from 0, but the sizes are actual size (cannot have size 0);

This is size

We would like to

- add more lockers dynamically to students
- Remove unused lockers
- Relocate lockers

A pain to do with Arrays!



Array vs ArrayList

- Array is a fixed length *Data Structure* (static), while ArrayList can be variable length *Collection Class* (dynamic)
- An ArrayList is an object and it stores the sequence of objects
- Array – both primitive types and objects
- ArrayList – **objects** ONLY
- ArrayList can be used to store different types of values
- Some performance difference – also applies to other java Collections, including Linklist – advance programming
- Operational difference e.g.
 - `anArray.length()` vs `anArrayList.size()`
 - Assignment for array vs insertion in ArrayList

Array vs ArrayList

Like you would make a new object

Array	ArrayList
<i>No need to import anything</i>	<code>import java.util.ArrayList;</code>
<code>int[] houseNumbers = new int[30];</code>	<code>List houseList = new ArrayList();</code>
<code>houseNumber[0] = 123;</code>	<code>houseList.add(new Integer(123));</code>
<code>??</code>	<code>houseList.remove(0);</code>
<code>houseNumber.length()</code>	<code>houseList.size();</code>
<code>int a=houseNumber[2];</code>	<code>houseList.get(0);</code>

Array uses `[]` for *size definition* and *index*

- Use a constructor method to create a new ArrayList
- Notice now we are using *methods()* to manipulate elements

Java Generics to create ArrayList

```
List houseList = new ArrayList();  
int a = (Integer) houseList.get(0);
```

- But what type is stored in houseList? Could it create problems?
- Use generics to provide **type (classes and interface) checks** at compile time.
- The type passed must be non-primitive
- You don't need to cast the retrieved elements

```
List<Integer> houseList = new ArrayList<Integer>();  
Integer a = houseList.get(0);
```

<http://docs.oracle.com/javase/tutorial/java/generics/>

Wrapper classes/object

- *A wrapper classes wraps a primitive type inside an object, which is needed when inserting them in an array list or other collection*
- *(note the capitals!)*

```
Integer intWrap = new Integer(10);  
Double doubleWrap = new Double(3.14159);  
Character charWrap = new Character('X');  
Boolean booleanWrap = new Boolean(true);
```

- *The value of wrapper objects cannot be changed*
- *Autoboxing and autounboxing - implicit conversion between the wrapper objects and primitive values*
- *Uses a constructor methods*

http://en.wikipedia.org/wiki/Primitive_wrapper_class

Value of wrappers

Wrapper objects have methods which return the **primitive values** stored inside them:

```
Double dWrap = new Double("33.33");  
double primitiveValue = dWrap.doubleValue();
```

Casting is also built in:

```
int intCast = dWrap.intValue();
```

And all objects have a `toString()` method

Filling the array list

```
ArrayList myList = new ArrayList();
```

```
Integer intWrap = new Integer(4);
```

```
Double doubleWrap = new Double(5.5);
```

```
Character charWrap = new Character('X');
```

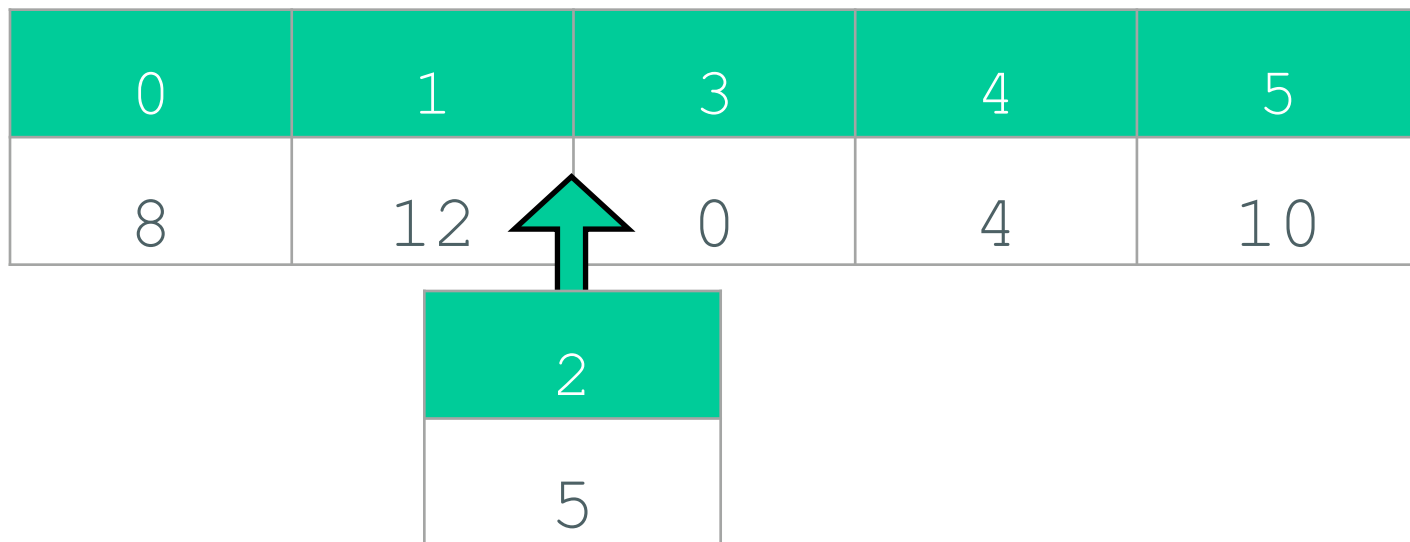
```
myList.add(intWrap);  
myList.add(doubleWrap);  
myList.add(charWrap);
```

It is valid to have different data types here, but mostly you won't do this.

Insert

If you use an index with the `add()` method, then the object is added at that index and all others are shifted:


```
myList.add(2, new Integer(5));
```



Remove

Use the `remove()` method to delete an element

```
myList.remove(2);
```

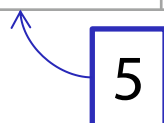
0	1		2	3
8	12	0	4	10

Accessing Element

```
myList.set(2, 5);  
int a = myList.get(2);
```

Conversion between int and Integer in
JAVA is called autoboxing or autounboxing

0	1	2	2	3
8	12	0	4	10



Searching in arrays

We want to find the first place in a `char` array where the letter `X` appears:

```
char[] myArray = new char[100];  
//Fill array with character values  
int i=0;  
boolean found;  
do {  
    found = (myArray[i]=='X') ? true : false;  
}  
while(found==false && i++ < myArray.length);
```

With an ArrayList

You can use the `indexOf()` method:

```
ArrayList myList = new ArrayList();  
//Fill myList with Character wrappers  
myList.indexOf(new Character('X'));
```

`indexOf()` returns the value -1 if the object it is searching for is not found

An example using ArrayList

1. Import class library

```
import java.util.ArrayList;
```

```
public class FruitExample {  
    public static void main(String[] args) {  
        ArrayList<String> alFruits = new ArrayList<String>();  
  
        alFruits.add("apple");  
        alFruits.add("orange");  
        alFruits.add("pear");  
  
        alFruits.remove("apple");  
        for (int i=0; i < alFruits.size(); i++) {  
            String sFruit = alFruits.get(i);  
            System.out.println(sFruit);  
        }  
    }  
}
```

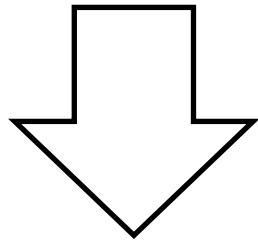
2. Create a new ArrayList object, ~ Use generics to define type

3. Fill ArrayList

4. Access and manipulating ArrayList

A “For each” loop

```
for(int i=0; i<myList.size(); i++) {  
    System.out.println(myList.get(i));  
}
```



The `toString()` method is called automatically

```
//For each object in myList  
for(Integer i : myList) {  
    System.out.println(i);  
}
```

Converting to arrays

The ArrayList class contains the method `toArray()`

You can only use this to create arrays of **objects**:

```
ArrayList myList = new ArrayList();  
//Fill array list  
Integer[] myArray = new Integer[myList.size()];  
myList.toArray(myArray);  
System.out.print(myArray[0]);
```

Primitive arrays

To read element values between a primitive array and an array list, it is best to use loops:

```
int[] myArray = {1,2,3,4,5};
ArrayList<Integer> myList = new ArrayList<Integer>();

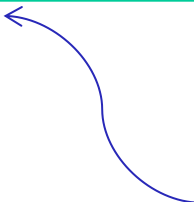
for(int i : myArray) myList.add(i);

for(int i=0; i<myArray.length; i++) {
    myArray[i] = myList.get(i);
}
```

A useful random example

```
import java.util.ArrayList;
import java.util.Random;
```

```
public class Raffle {
    //returns array of 10 raffle winners from list of names
    public static String[] getNames(String[] names) {
        ArrayList<String> list = new ArrayList<String>();
        Random gen = new Random();
        for(String player : names) {
            list.add(player);
        }
        String[] winners = new String[10];
        for(int i = 0; i<10; i++) {
            winners[i] = list.remove(gen.nextInt(list.size()));
        }
        return winners;
    }
}
```



Get the element also remove it from the list



```
import java.util.Arrays;

public class RaffleTest {
    public static void main(String[] args) {
        String[] names = {"Jacqueline Tia", "Hanan Abdi", "Tila Q", ... };
        String[] winners = Raffle.getNames(names); //because it was static, no
        object instance.
        System.out.println(Arrays.toString(winners));
    }
}
```

Example

Write a program that reads in comma separated data defining book objects from a file given at command line, creates an array list called `library` and adds new Book objects to the array list for each line defined in the file.

Loop through the array list and print out each book.

We will look at file handling next week.

```
import java.util.ArrayList;
import java.util.Collections;
```

```
public class MakeLibrary {
    public static void main(String[] args) {
```

```
        ArrayList<Book> library = new ArrayList<Book>();
```

```
        fillLibrary(args[0], library);
```

```
        //print out all books in library
```

```
        for(Book b : library) {
            System.out.println(b);
        }
```

```
        Collections.sort(library, Book.RATING);
```

```
        System.out.println("");
```

```
        for(Book b : library) {
            System.out.println(b);
```

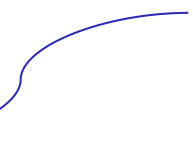
```
        }
```

```
    }
```

ArrayList of “Book” class type



“For each” available
version >JAVA 5



“Collections” class library

```
public static void fillLibrary(String fileName, ArrayList<Book> library) {  
    FileInput in = new FileInput(fileName); //we will look at fileInput next week!  
    String[] line;  
    String title;  
    String author;  
    int pubYear;  
    int rating;  
    while(!in.eof()) {  
        line = in.readString().split(",");  
        if(line.length == 4) {  
            //find data and add new book object to library  
            title = line[0];  
            author = line[1];  
            pubYear = Integer.parseInt(line[2]);  
            rating = Integer.parseInt(line[3]);  
            library.add(new Book(title, author, pubYear, rating));  
        }  
    }  
}
```

Java.util.Arrays class library

- This class library contains various methods for manipulating arrays (such as sorting and searching). This class also contains a static method that allows arrays to be viewed as lists.
- `.equals(Object[] a, Object[] a2)`
Returns true if the two specified arrays of Objects are *equal* to one another.
- `.Sort` and `.toString` are also very useful.

Sorting

You can also sort objects in array lists using the `sort` method in the `Collections` class (import `java.util.Collections`)

Objects need to know how to compare themselves to each other

Comparing objects

Strings, Dates and numerical objects (Integer, Double, Float etc.) are compared in the normal way

You can help the sort method by defining comparators inside your object

Sorting by field

We can sort the library array list by each field of the object in turn:

```
Collections.sort(library, Book.AUTHOR);  
Collections.sort(library, Book.TITLE);  
Collections.sort(library, Book.PUBYEAR);  
Collections.sort(library, Book.RATING);
```


Override toString()

toString() will just print out the hashCode of the object from an ArrayList.

We can override this to give a better, custom made description:

```
@Override public String toString() {  
    return title + ", "  
        + author + ", "  
        + pubYear + ", "  
        + rating + "." ;  
}
```

MORE ON STRING

String

- String is a Class

1) Creating String objects using constructor

```
String words2 = new String("I am a string");
```

2) Using String Literals directly

```
String someWords = "I am a string";
```

- There is a subtle difference in the memory addressing -

Using “=” with strings

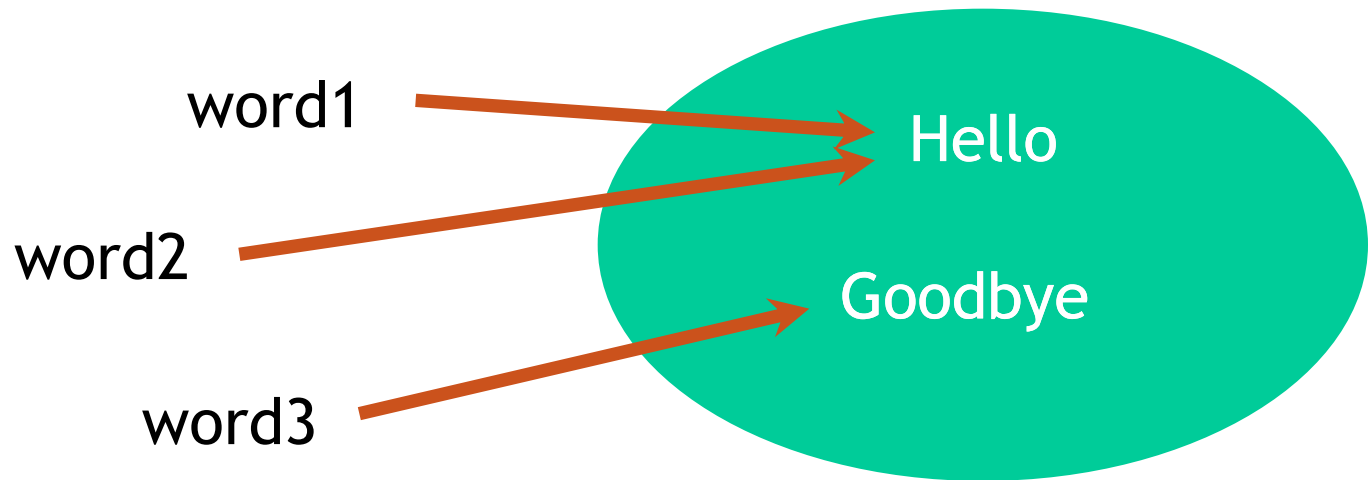
The equals sign “=”, when used with a string object, just makes a link between the object name and the address of a string literal

String literals are all stored in the string literal pool

The string literal pool keeps one copy of each string literal used so that memory usage is more efficient

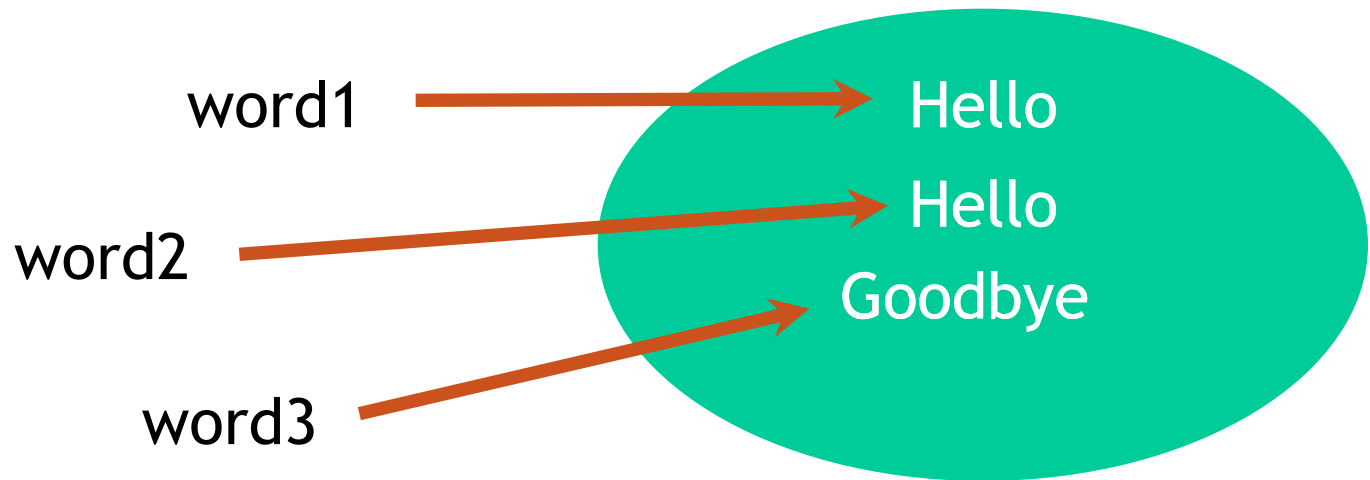
The string literal pool

```
String word1, word2, word3;  
word1 = "Hello";  
word2 = "Hello";  
word3 = "Goodbye";
```



Using the `new` command

```
String word1, word2, word3;  
word1 = "Hello";  
word2 = new String("Hello");  
word3 = "Goodbye";
```



```
class JackAndJill {  
    public static void main(String args[]) {  
        String s1 = new String("Jack went up the hill.");  
        String s2 = new String("Jack went up the hill.");  
        if ( s1 == s2 ) {  
            System.out.println("The strings are the same.");  
        }  
        else if ( s1 != s2 ) {  
            System.out.println("The strings are not the same.");  
        }  
    }  
}
```

!! The result is the strings are not the same!!

String.equals(String)

```
class JackAndJill {  
    public static void main(String args[]) {  
        String s1 = new String("Jack went up the hill.");  
        String s2 = new String("Jack went up the hill.");  
        if ( s1.equals(s2) ) {  
            System.out.println("The strings are the same.");  
        }  
        else {  
            System.out.println("The strings are not the  
same.");  
        }  
    }  
}
```


What about “==” ?

Two string objects are only equal (==) if they point to the same string literal in the pool

If you create two objects using the `new` command, then `word1==word2` is always false

To compare the sequence of characters stored by a string, use the `equals()` method instead:

```
word1.equals(word2) //returns true
```

The moral of the story

- 1) If creating strings from literals, always use “=” instead of the `new` command, to save memory
- 2) If comparing the value of strings, always use the `equals()` method instead of “==”

An array of characters?

In many ways, a string is like an array of characters, with some additional methods added

You can access characters in a string by index:

```
String someWords = "I am a string";
```

```
//Use the charAt() method
```

```
char myChar = someWords.charAt(3);
```

```
//myChar = 'm'
```

Conversion to array

You can use a character array as an argument in the constructor method to create a new string, and you can use the `toCharArray()` method of a string object to return an array of characters:

```
char[] myArray = { 'H', 'e', 'l', 'l', 'o' };  
String hello = new String(myArray);  
char[] myArray2 = hello.toCharArray();
```

Get and set method?

No! Strings are immutable: once you have created them, you cannot change their elements

Instead, you should assign them to a new object

```
String mistake = "Hillo";  
char[] fix = mistake.toCharArray();  
fix[1] = 'e';  
String hello = new String(fix);
```

Additional characters

Strings can accept any unicode character, but whether or not your computer can handle these will depend on what fonts you have installed

There are also some useful formatting characters:

`\n` : new line

`\t` : tab character

`\"` : escape quotes

`\'` : single quote

`\b` : backspace

`\r` : return

`\\` : escape backslash

Additional characters 2

You can even use unicode values for variable and method names:

```
String message = "我超喜歡電腦程式";  
for(char 漢字 : message) {  
    System.out.println(漢字);  
}
```

Be careful: this can make sharing code more difficult and not all IDEs will allow unicode characters!

StringBuffer “append” is better than + with strings

We need to convert the array of string objects into a single string object - you can use += to concatenate each string together

```
myString += args[index];
```

This will be a little inefficient: adding a number of Strings by + creates a new String object each time it loops. Look up the `StringBuffer` class if you want to learn a better way

Slide practise

Typical programming question (e.g. at interviews)

Write a program that accepts a message in letters and spaces and shifts each letter by a fixed amount through the alphabet

The ASCII values representing `chars` are:

A = 65, B = 66, C = 67, ... , Z = 90

a = 97, b = 98, c = 99, ..., z = 122

' ' = 32

```
String original = "This is my message";
```

```
char[] array = original.toCharArray();
```

```
int shift = 1;
```

```
for(int i=0; i<array.length; i++) {
```

```
    int code = array[i];
```

```
    if(code >= 'A' && code <= 'Z') {
```

```
        //Upper case
```

```
        array[i] = (char)('A' + (code-'A'+shift)%26);
```

```
    }
```

```
    else if (code >= 'a' && code <= 'z') {
```

```
        array[i] = (char)('a' + (code-'a'+shift)%26);
```

```
    }
```

```
    else array[i] = ' ';
```

```
}
```

```
String coded = new String(array);
```

**Manipulate characters
using their ASCII code**



Converting Strings to int/ double, etc

- The syntax uses the static **Integer.valueOf(String s)** and **intValue()** methods from the `java.lang.Integer` class.
- To convert the String "22" into the int 22 you would write

```
int i = Integer.valueOf("22").intValue();
```
- Others
 - `long l = Long.valueOf("22").longValue();`
 - `double x = Double.valueOf("22.5").doubleValue();`
 - `float y = Float.valueOf("22.5").floatValue();`

The `args` array

Slide practice 4.3 : $E=MC^2$

- Write a program called Einstein that takes in a value for Mass at the Command line prompt via args, and calculates and outputs the energy in Joules.
- The value for c (the speed of light) is given as
- `double c = 2.998E8; // meters/second`
- `String[] args` **part of the main method?**

```
public static void main (String args[])
```

- This allows you to give string arguments to the program when you run it:

```
% javac iceCreamCone.java
```

```
% java iceCreamCone Chocolate Vanilla Pistachio
```

```
class Einstein {  
    public static void main (String args[]) {  
        double c = 2.998E8; // meters/second  
        double mass = Double.valueOf(args[0]).doubleValue();  
        //need to convert chars into double value.  
        double E = mass * c * c;  
        System.out.println(E + " Joules");  
    }  
}
```

A few more useful String methods

```
String myString = "Some Useful Things";  
myString.toLowerCase();  
//returns "some useful things"  
myString.toUpperCase();  
//returns "SOME USEFUL THINGS"  
myString.substring(0, 5);  
//returns "Some U";  
myString.split(" ");  
//returns {"Some", "Useful", "Things"}
```

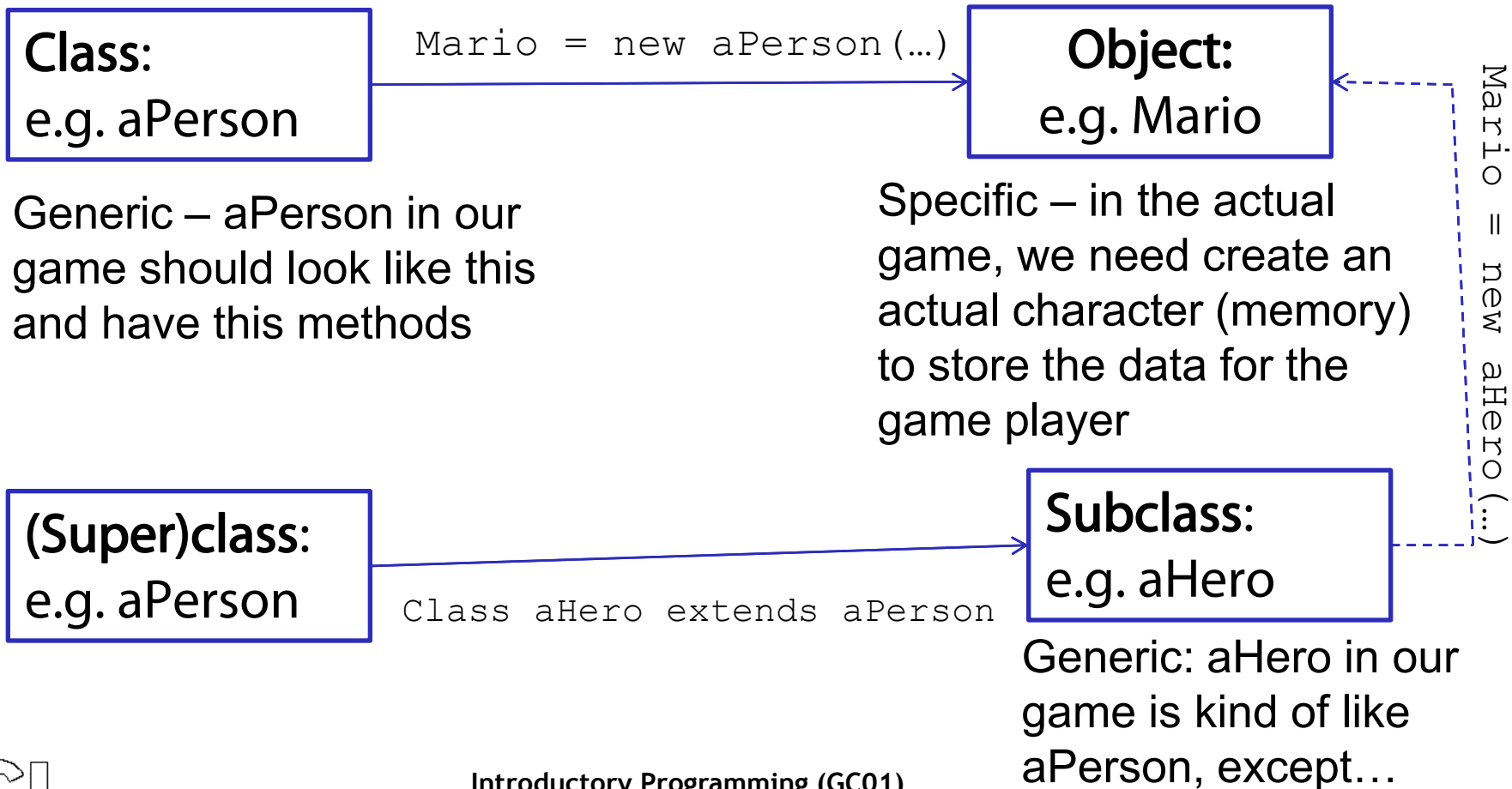
INHERITANCE AND INTERFACE

Java structure types

1. Main class
2. Library / static class
3. Object class
- 4. Inherited class**

Object class and Object Overview

Super-Mario example



Beginners guide to Inheritance

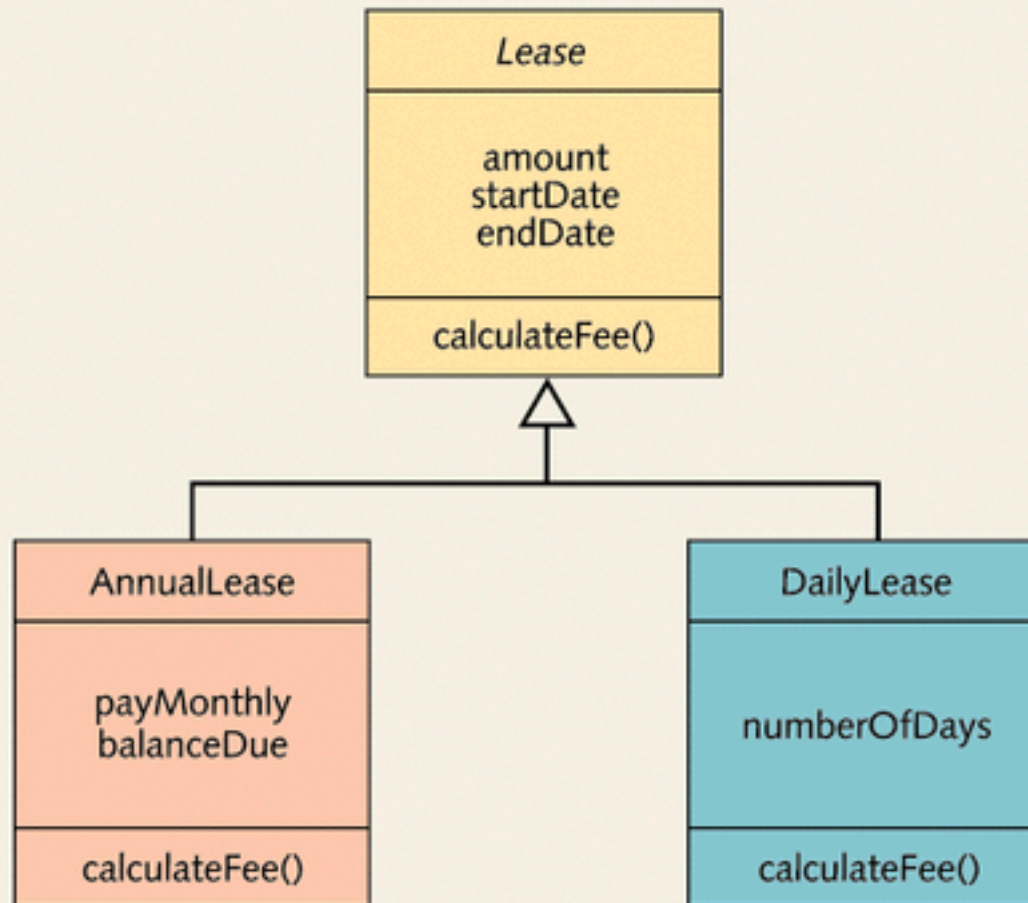
- We need to cover a few key concepts on objects before continuing on.



Extends

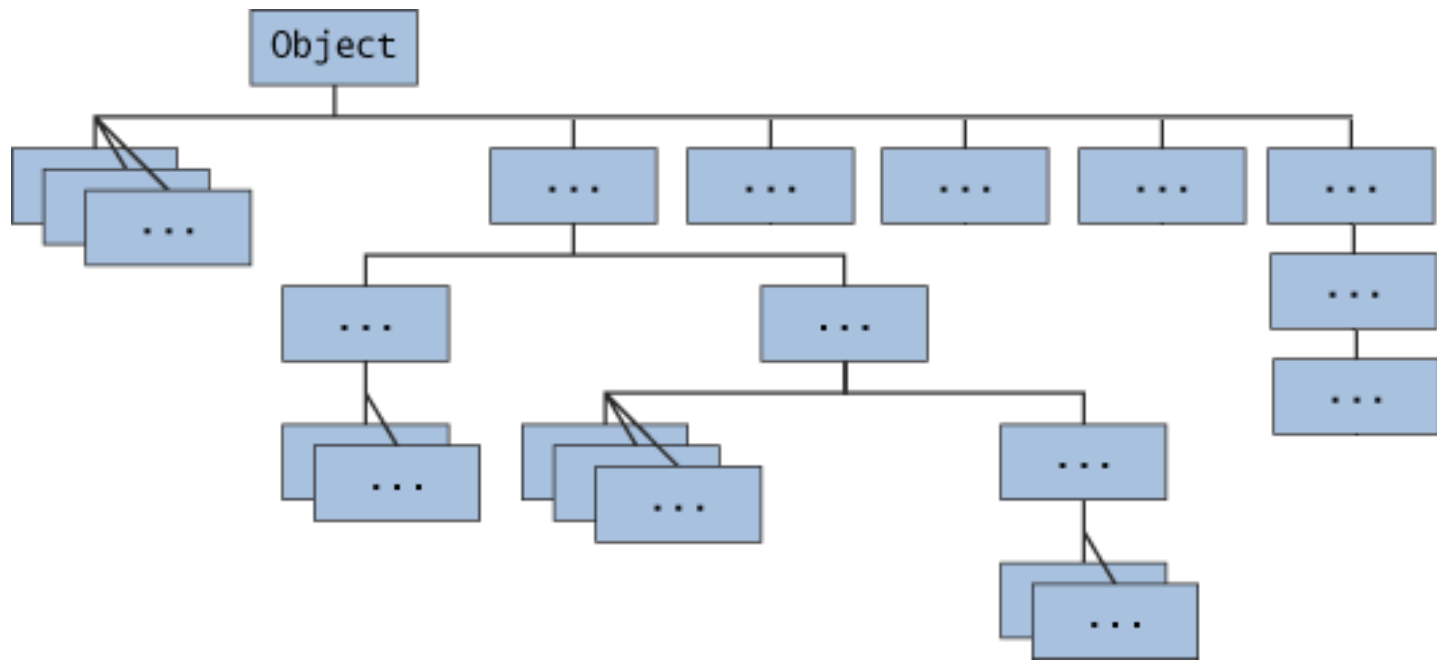
```
class Fruit {  
    //... }  
class Apple extends Fruit {  
    //... }
```

- class Apple is related to class Fruit by inheritance, because Apple extends Fruit.
- In this example, Fruit is called the *superclass* and Apple is the *subclass*.
- **Make sure inheritance models the “*is-a*” relationship**
- inheritance should be used only when a subclass *is-a* superclass. In the previous example, an Apple is-a Fruit



Object class is the Godfather.

- Except Object, which has no superclass, every class has one and only one **direct** superclass (single inheritance).
- In the absence of any other explicit superclass, every class is implicitly a **subclass of Object**. This is how you have access to Object class methods like toString() and clone().
- Classes can be derived from classes that are derived from classes that are derived from classes. Such a class is said to be *descended* from all the classes in the inheritance chain stretching back to Object.



Why inheritance

- No need to write your code from scratch/zero. Start from something already there
- A subclass inherits all the *members* (fields, methods, and nested classes) from its superclass. We often do this to make more detailed classes from a basic class.
- `@Override` if you want a method to be different

Override Object.toString()

```
class aClass {  
    private int aNum;  
  
    public aClass(int a) {  
        aNum = a;  
    }  
  
    public int getNum() {  
        return aNum;  
    }  
  
    @Override  
    public String toString() {  
        //return (getClass().getName() + '@' + Integer.toHexString(hashCode()));  
        return (getClass().getName() + " Object number " + aNum);  
    }  
}
```

Super keyword

For a class Student that extends a class Person..

```
//student constructor
public Student(String initialName, int initialStudentNumber)
{
    super(initialName); //calls the base constructor, in this
    case with a parameter
    studentNumber = initialStudentNumber;
}
```

Note: You do not use the name of the constructor;
e.g. you do *not* use Person(initialName).

Super comes first

- It must always be the **first action** taken in an inherited constructor definition.
 - You cannot use it later in the definition of a constructor.
- In fact, if you do not include a call to the base-class constructor, then Java will automatically include a call to the default constructor of the base class as the first action of any constructor for a derived class.
- You use super, especially to **select which constructor** you want from the base class.

Box example

```
class Box {
```

```
    double width;  
    double height;  
    double depth;
```

```
    Box() {  
    }
```

```
    Box(double w, double h, double d) {  
        width = w;  
        height = h;  
        depth = d;  
    }
```

```
    void getVolume() {  
        System.out.println("Volume is : " + width * height * depth);  
    }  
}
```

Have package-level access (default) if access modifier is not specified

Box example – cont'

```
public class MatchBox extends Box {
```

```
    double weight;
```

```
    MatchBox() {
```

```
    }
```

```
    MatchBox(double w, double h, double d, double m) {
```

```
        super(w, h, d);
```

```
        weight = m;
```

```
    }
```

```
    public static void main(String args[]) {
```

```
        MatchBox mb1 = new MatchBox(10, 10, 10, 10);
```

```
        mb1.getVolume();
```

```
        System.out.println("width of MatchBox 1 is " + mb1.width);
```

```
        System.out.println("height of MatchBox 1 is " + mb1.height);
```

```
        System.out.println("depth of MatchBox 1 is " + mb1.depth);
```

```
        System.out.println("weight of MatchBox 1 is " + mb1.weight);
```

```
    }
```

```
}
```

Inherited method



“this” in inheritance

- “this” refer to the **current object**.

```
public Point(int x, int y) {  
    this.x = x; this.y = y;  
}
```

- “this” with a Constructor

It refers to the **current inherited subclass**, not the superclass. The call is to a constructor of the same class and not a call to a constructor for the base class.

```
public Rectangle() {  
    this(0, 0, 1, 1);  
}  
  
public Rectangle(int width, int height) {  
    this(0, 0, width, height);  
}  
  
public Rectangle(int x, int y, int width, int height) {  
    this.x = x; this.y = y; this.width = width;  
    this.height = height;  
}
```

Rules

- A subclass can extend only one superclass.
- A subclass inherits all of the *public* and *protected* members of its parent, no matter what package the subclass is in.
- If the subclass is in the same package as its parent, it also inherits the *package-private (no modifier)* members of the parent.
- A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods for accessing its private fields, these can also be used by the subclass. Hence the need for designing get and set methods.
- **Private methods Are Not Inherited.**

“Final” keyword

The “final” Modifier

- If you want to specify that a method definition cannot be overridden with a new definition in a derived class, then you can do so by adding the final modifier to the method heading, as in the following sample heading:
- `public final void specialMethod() { ...`
 - You are not very likely to need this modifier, but you are likely to see it in the specification of some methods in standard Java libraries.
- An entire class can be declared final, in which case you cannot use it as base class to derive any other class from it – cannot “extends” it
- Final **field** cannot be changed once its initiated. E.g. constants

Two more types

1. Main class
2. Library / static class
3. Object class
4. Inherited class

5. Interface type
6. Abstract class type

Interface type

- an *interface* is a **reference type**,
 - can contain only constants, method signatures , static methods, default methods and nested types.
 - Method bodies exist ONLY for default methods and static methods
 - a class without any functionality, bare bones but with all of the necessary details to give it a structure.
- Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces.
- Imagine you are defining the communication protocols for two classes, you want to **how** they will communicate with each other, not exactly **what** they talk about.

An interface example

```
public interface OperateCar {  
    // constant declarations, if any  
  
    // method signatures  
    int turn(String direction, double radius, double  
        startSpeed, double endSpeed);  
    int signalTurn(String direction, boolean signalOn);  
  
    .....  
    // more method signatures  
}
```

Implement an interface

- To use an interface, you write a class that implements the interface. When an instantiable class implements an interface, it provides a method body for each of the methods declared in the interface. For example,

```
public class OperateBMW implements OperateCar {  
  
    // the OperateCar method signatures, with implementation --  
    // for example:  
    int signalTurn(String direction, boolean signalOn) {  
        //code to turn BMW's LEFT turn indicator lights on  
        //code to turn BMW's LEFT turn indicator lights off  
        //code to turn BMW's RIGHT turn indicator lights on  
        //code to turn BMW's RIGHT turn indicator lights off  
    }  
}
```

Interface types

- Useful situations for writing “interface”
 - Build software in teams
 - Defines how pieces of software interacts, like “contracts”
 - Define public Application Programming Interface (API)
 - » E.g. a package of digital image processing methods that are sold to companies making end-user graphics programs
- E.g. Define a standard interface between a vehicle and a GPS works together. E.g. getLocation(), setLocation(), getTraffic(), etc....
 - These are signature, so the vehicle company does not need to know the detail of how GPS companies calculate these.
 - Different GPS companies have their “proprietary” way of calculating the methods.
 - They just need to complies with the standard

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

java.util

Class ArrayList<E>

java.lang.Object

java.util.AbstractCollection<E>

java.util.AbstractList<E>

java.util.ArrayList<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

```
public class ArrayList<E>
```

```
extends AbstractList<E>
```

```
implements List<E>, RandomAccess, Cloneable, Serializable
```

Implementing List interface

```
List houseList = new ArrayList();
```

- Why List not ArrayList?
- ArrayList and LinkedList both used for storing list of objects
- Performance and operational differences
- If you stick with declaring houseList with a List interface, you could change your declaration from one to the other later on, without altering your code.

ABSTRACT CLASS AND METHODS

Abstract class type

- An ***abstract class*** is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed (using extends).
- If a class includes one or more abstract methods, the class itself must be declared abstract, as in:

```
public abstract class GraphicObject {  
    // declare fields  
    // declare non-abstract methods  
    abstract void draw();  
}
```

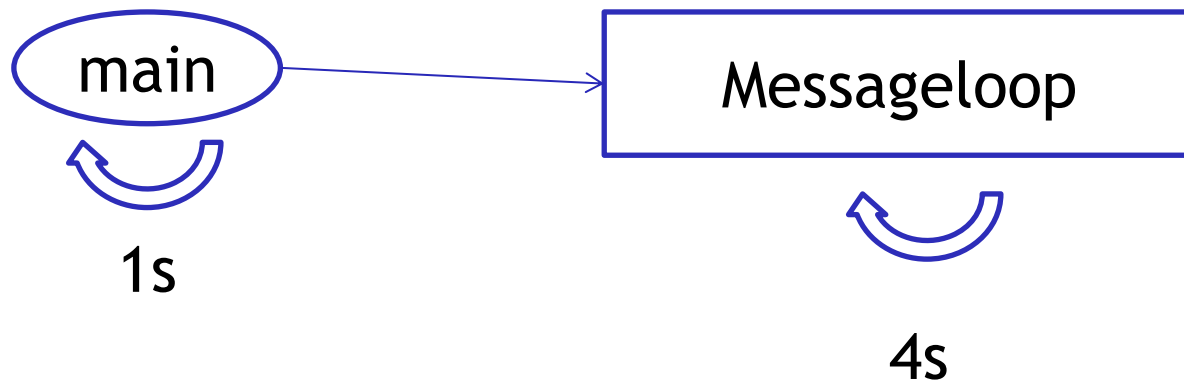
Abstract method without
implementation

Abstract vs Interface

- **Note:** All of the methods in an *interface* are *implicitly* abstract, so the abstract modifier is not used with interface methods.
- Unlike interfaces, abstract classes **can** contain fields that are not static and final, and they **can** contain implemented methods.
- Such abstract classes are similar to interfaces, except that they provide a partial implementation, leaving it to subclasses to complete the implementation. If an abstract class contains *only* abstract method declarations, it should be declared as an interface instead.

A few words on threading

- Processor allows several “tasks” to run “concurrently”, to be active at the same time – sharing resources and processor time
- Threads are light weight “thread”
- E.g. GUI – listening to user inputs + usb communication with some external device + processing some results
- Processor **schedule** and **prioritise** tasks
- <http://docs.oracle.com/javase/tutorial/essential/concurrency/simple.html>
- <http://docs.oracle.com/javase/tutorial/uiswing/concurrency/initial.html>



InvokeLater

```
SwingUtilities.invokeLater(new Runnable() {  
    public void run() {  
        createAndShowGUI();  
    }  
});
```

Throw exceptions

- When using some methods, you are required to throw/catch an exception object
- Try & catch – to handle the situation when something doesn't work
 - E.g. when you try to open a file and it doesn't exist
 - `java.io.FileNotFoundException`
- Can throw more than 1 exceptions
- <http://docs.oracle.com/javase/tutorial/essential/TOC.html>

Read up on these topics.

OVERRIDING, OVERLOADING, MULTIPLE IMPLEMENTS – SINGLE EXTENDS..

HTML OVERVIEW

Quick introduction to HTML

- Out of interest, lets look at how to save text to a HTML file.
- All HTML tags are just formatting tags, and most need to be closed to work with that section.
- `<html>` starts a html document, `</html>` ends a html document.
- `<p>` for paragraph, `</p>` for closing the paragraph.
- Write some text in between e.g. `<p> hello </p>`
- Save it as index.html (or any name , ending .html)
- Open it in a browser.

Apps use HTML

- Many pages (screens/Activities) of an app can be designed in HTML. It makes it much easier to port complex designed screens across different platforms.
- Showing these pages can be done by putting it into a HTML rendering frame – for example in Java, a JTextPane from Java Swing can be used.
- In Android – a WebView class can also do the same.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
■ import java.net.*;
public class TestShowPage {
    public static void main(String args[]) {
        JTextPane tp = new JTextPane();
        JScrollPane js = new JScrollPane();
        js.getViewport().add(tp);
        JFrame jf = new JFrame();
        jf.getContentPane().add(js);
        jf.pack();
        jf.setSize(400,500);
        jf.setVisible(true);
        try {
            URL url = new URL("http://www.cs.ucl.ac.uk");
            tp.setPage(url);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Example from rgagnon.com



Web centric

- We live in a web centric world.
- A lot of finance and internet based companies want to know about programs that create web content faster.
- A good theme to cover with what we have learnt so far.
- AJAX and jQuery are big in the field, they use java like notation.

HTML is not programming

- We cant use HTML tags for making decisions and operating logic, such as And, Or and Not.
- HTML is formatting, it also includes some browser designed form elements for us to generate web forms.
- We use JavaScript – a very java like language – to be able to do programming in a web page and interact with forms etc.
- We'll look at JavaScript later in the course, but for now, a quick introduction to HTML tags that may prove useful soon.

A few HTML tags – brief summary

- `<html></html>` Creates an HTML document
- `<head></head>` Sets off the title and other information that isn't displayed on the Web page itself
- `<body></body>` Sets off the visible portion of the document
- `<title></title>` Puts the name of the document in the title bar
- `<h1></h1>` Creates the largest headline, try h2, h3 to see sizes.
- `<h6></h6>` Creates the smallest headline
- `<p></p>` Creates a new paragraph
- `<p align=?>` Aligns a paragraph to the left, right, or center
- `
` Inserts a line break
- `` Creates bold text
- `<i></i>` Creates italic text
- `` Creates a hyperlink
- `` Creates a mailto link
- `` Creates a target location within a document
- `` Precedes each list item and adds a number
- `` Creates a bulleted list

A few more!

- `` Adds an image from a relative folder
- `<hr>` Inserts a horizontal rule
- `<table></table>` Creates a table
- `<tr></tr>` Sets off each row in a table
- `<td></td>` Sets off each cell in a row
- `<th></th>` Sets off the table header (a normal cell with bold, centered text)
- `<form></form>` Creates a forms
- `<select name="NAME"></select>` Creates a pulldown menu
- `<option>` Sets a menu item
- `<textarea name="NAME" cols=40 rows=8></textarea>`
Creates a HTML text box area - columns set the width; rows set the height.
- `<input type="checkbox" name="NAME">`
Creates a checkbox - text follows tag.
- `<input type="radio" name="NAME" value="x">`
Creates a radio button - text follows tag
- `<input type="text" name="foo" size=20>`
Creates a one-line text area - size sets length in characters.
- `<input type="submit" value="NAME">`
Creates a Submit button
- `<input type="image" border=0 name="NAME" src="name.gif">`
Creates a Submit button using an image
- `<input type="reset">`
Creates a Reset button

The History of HTML/XHTML

- 1992 – HTML first defined
- 1994 – HTML 2.0
- 1995 – Netscape specific non-standard HTML
- 1996 – HTML 3.2, compromise version
- 1997 – HTML 4.0, separates content from presentation
- 1998 – XML standard for writing Web languages
- 2000 – XHTML 1.0, XML compliant HTML
- 2002 – XHTML 2.0

XHTML

- XHTML is a version of HTML modified to conform to the XML standard
- Designed to separate content from presentation
 - Content in XHTML
 - Presentation controlled by Cascading Style Sheets (CSS)
- Extensible – Additional elements can be defined
- XML Compatible – Other XML based languages can be embedded in XHTML documents
- Like a programming language
 - Specific syntax to use
 - Validators help you get it right

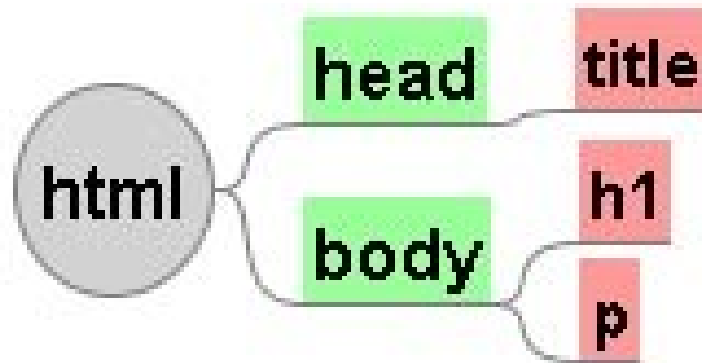
XHTML Differences

- Case is significant
- All elements must have begin tags and end tags `<p>Hello</p>`
- Empty elements contain their own end tag
`
`
- Attribute values must be enclosed in quotation marks
- More specifics available at <http://www.w3.org/TR/xhtml1/#diffs>

A Simple XHTML Webpage (save as .html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>
      My Home Page
    </title>
  </head>
  <body>
    <h1>My Home Page </h1>
    <p>
      Welcome to my home page
    </p>
  </body>
</html>
```

Hierarchical Structure



Well formed xhtml forms a hierarchy

The DOCTYPE Statement

- Declares the specific version of HTML or XHTML being used on the page
- Used by the browser to decide how to process the page
- Three types
 - Transitional - Forgiving
 - Strict – Requires adherence to standards
 - Frameset – Use if page has frames
- Always first in file

Strict DOCTYPE

- Enter exactly as below

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

- Using Strict encourages standards based coding
 - Validators will flag logical errors in your methods

XHTML Elements

- Same as HTML constructs – however, must be complete with three parts
 - Begin tag, which can contain attributes
 - Contents
 - End tag
- Example:
<p id="intro">Welcome</p>
- W3schools specifications for <p>
http://www.w3schools.com/tags/tag_p.asp

XHTML Attributes

- Always only used in the element begin tag
- Three types
 - Optional attributes: Varies with element type
 - Standard attributes: id, class, title, style, dir, lang, xml:lang
 - Useful Event attributes: onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
 - » Used in scripting!!

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>example of OnMouseOver in XHTML</title>
</head>
<body>
<input type="button" onmouseover="JavaScript: alert( 'onmouseover event' )" value="Move The Mouse Over This Button!" />
</body>
</html>
```