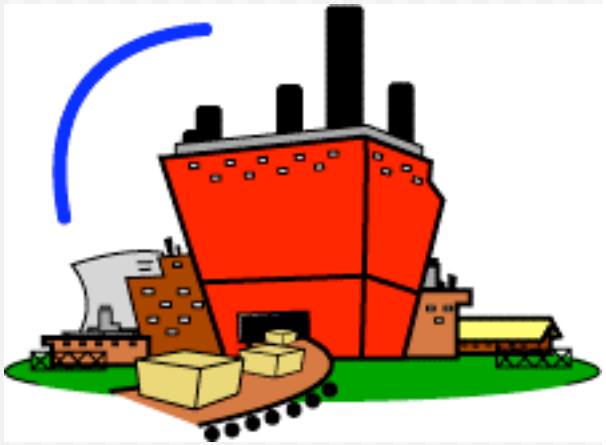
Computer Science 101 SS C

Lecture 15

Classes 1



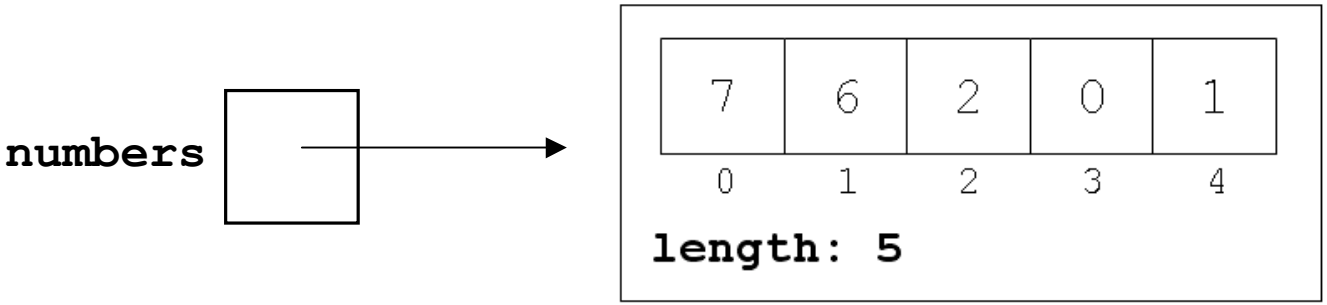
Summary

We have introduced the use of arrays for both primitive types and for objects.

An array:

- holds a sequence of variables of the ***same type***
- each ***element*** in the array is associated with an ***index*** value
- index values start at 0
- every array has a ***length*** field which stores the number of elements in the array
- ***arrays are objects***

int [ ] numbers = {7, 6, 2, 0, 1};



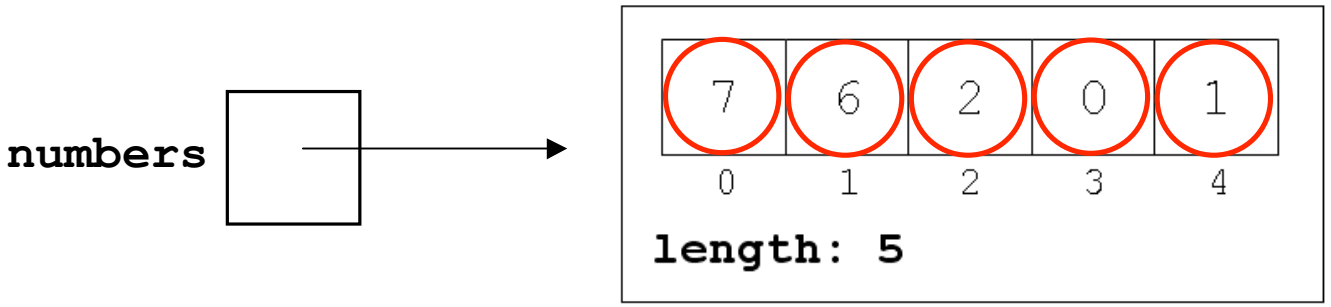
Summary

We can access every element in an array systematically using a loop:

this is how we access the `length` field of an array

This is commonly done with a `for` loop like:

```
for (int i = 0; i < numbers.length; i++) {
      System.out.println(numbers[i]);
}
```



Summary

We discussed how aliasing works with object variables, and how it can lead to confusing code.

We also showed how arrays can be passed as parameters or as return values.

Lastly we examined how object parameters are a form of aliasing and saw how an understanding of aliasing allows us to predict the effect of “updating” object parameters.

Contents

Java is an object oriented programming language
Representing real world objects in programs
Structure of a class definition
Instance variables
Constructors
Instance methods
Common Errors

Coursebook: pages **§14.1 - §14.4.1**

Java is Object Oriented

The world is made up of real world objects
e.g. students, dogs, cars, cats, books.

Objects are the things our programs deal with.

State and behaviour

Each real world object has:

state - information stored about an object.

and

behaviour - functionality of the object i.e what can you do with that object.

Example 1 - state and behaviour

Consider a system managing university students. A student object has:

state e.g.

id, name, age, contact number, address, stage, completed courses, current courses, faculty, …

behaviour e.g.

add a new course, change contact number, change address, set faculty, …

Example 2 - state and behaviour

Consider a system managing university courses. A course object has:

state e.g.

id, name, faculty, list of current students, stage number, list of lecture times, semester offered, campus, test date, exam date …

behaviour e.g.

add/remove student to course,
change test date, change lecture time,
set semester offered …

Ex01 - state and behaviour

Consider a system managing university lecture rooms. A lecture room object has:

state e.g.

behaviour e.g.

Software objects - state and behaviour

A software object stores its state in one or more **variables**

A software object implements its behaviour with **methods**

Software objects - state and behaviour

Every object is a bundle of variables and related methods.

Note: The term, object, is the same as the term, instance.

Ex02 - String objects

In programs you have used **`String`** objects. What is the state and behaviour of a String object?

```
String word1 = new String( "object" );
String word2 = new String( "instance" );
word2 = word2.toUpperCase();
String letter = word1.substring(0,1);
int length = word1.length();
```

Java class

A class is the structure we use to define a category of objects. A class defines the state and behaviour of a category of objects.

A class corresponds to a Java type.

Class and objects

A class is a template for creating objects. Classes are the blueprint defining the state and the behaviour which any instance of that class will have.

Analogies for class and object:

Cookie cutter and cookies.

Factory mold and products produced from that mold.

Form stamp and filled out forms.

Java class definition

Java class definitions have the following structure:

```
public class ClassName {
    instance variables

    constructors

    instance methods
}
```

Instance variables

Instance variables allow us to define what information each instance of the class will store.

These variables are defined under the class header OUTSIDE any method definition.

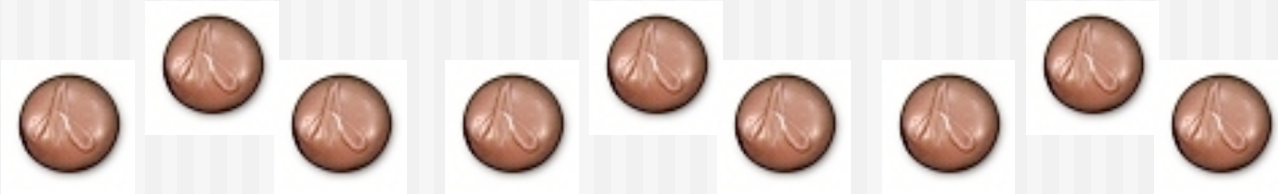
The scope of an instance variable is the whole class.

(In CompSci 101 instance variables which we define always have the modifier **`private`**.)

Example - instance variables

Consider defining a class to represent individual chocolates. The instance variables represent the information which all instances of the **`Chocolate`** class will contain.

```
public class Chocolate {
   private int code;
   private String description;
   private double price;
}
```



What is a constructor?

Constructors are special methods defined inside a class. Constructors are called whenever a new instance of the class is required.

Continuing with the analogy of a class being like the mold in a factory. Whenever we want to create a new object from the mold, we make a call to the constructor telling the factory we want to create a new object from that mold.

Constructor - rules

- The name of the constructor is the same as the class name.
- The constructor does not have a return type in the header.
- Constructor definitions are nearly always preceded by the modifier **`public.`**
- Constructors are often used to initialise the instance variables.
- A class can contain more than one constructor.

Example 4 - constructor

```
public class Chocolate{
   private int code;
   private String description;
   private double price;

   public Chocolate(int c,String desc,double p){
      code = c;
      description = desc;
      price = p;
   }
}
```

Instance methods

Instance methods are used to define the behaviour of an object i.e. what operations are possible on objects of the class.

These methods are defined inside the class.

We define methods the same way as we have done in all our java programs.

Instance methods - which ones?

What do we want to be able to do with an instance of this class?

`getPrice()`: get the price of a single **`Chocolate`** object.

`pricePer20()`: get the price for 20 **`Chocolate`** objects.

`setDescription()`: change the description of the **`Chocolate`** object.

Example 5 - instance methods

```
public class Chocolate {
   private int code;
   private String description;
   private double price;
   public Chocolate( int c, String desc, double p ) {
      code = c;
      description = desc;
      price = p;
   }
   public double getPrice() {
        return price;
   }
   public double getPrice20() {
        return price * 20 * 0.8;
   }
   public void setDescription( String desc ) {
        description = desc;
   }
}
```

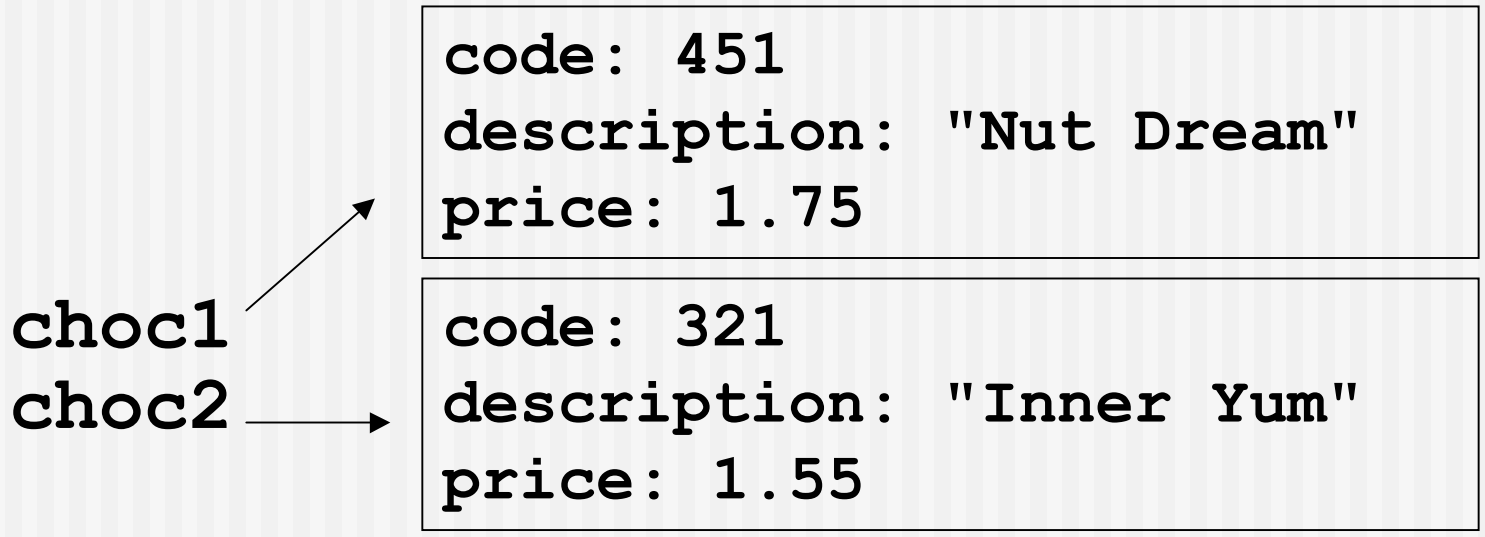
`Chocolate` class - creating instances

We are now able to use the `Chocolate` class definition to create as many new instances as are needed.

```
public class ProgramClassL14 {
 public void start() {
   Chocolate choc1, choc2;
   choc1 = new Chocolate(451,"Nut Dream",1.75);
   choc2 = new Chocolate(321,"Inner Yum",1.55);
 }
}
```

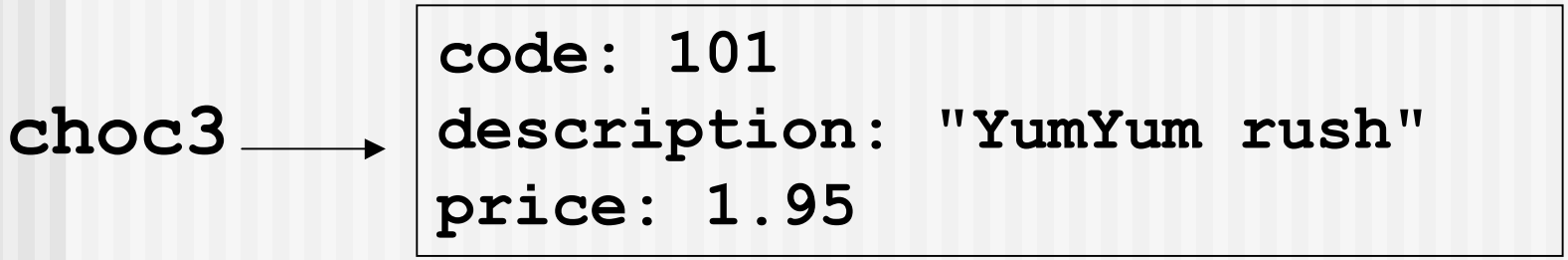
Visualising `Chocolate` class objects

`choc1` and `choc2`, created in the `start()` method (previous slide) have their own copy of ALL the instance variables (`code`, `description`, `price`). The `choc1` and `choc2` objects can be visualised:



Ex03 - creating instances

Write code which creates the instance shown below:

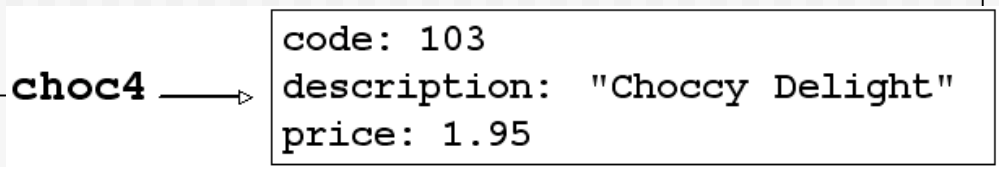



```
public class ProgramClassL14 {
 public void start() {
   Chocolate choc3;


 }
}
```

Dot notation

Once an object has been created, we can call the instance methods using that object (dot notation):

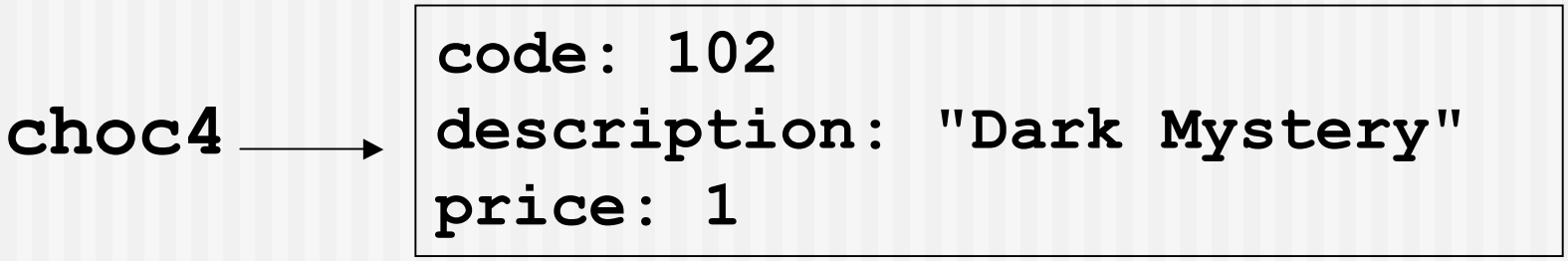
```
public class ProgramClassL14 {
 public void start() {
  Chocolate choc4;
  choc4 = new Chocolate( 103, "Nuts", 1.95 );
  choc4.setDescription( "Choccy Delight" );

  double price1 = choc4.getPrice();

  double price20 = choc4.getPrice20();

  System.out.println( "$" + price1 + ", $"+ price20 );
 }
}
```



Ex04 -calling instance methods

What is the output?

```
public class ProgramClassL14 {
 public void start() {
  Chocolate choc4;
  choc4 = new Chocolate( 102, "Divine",1 );
  choc4.setDescription( "Dark Mystery" );
  double price1 = choc4.getPrice();
  double price20 = choc4.getPrice20();
  System.out.println( "$" + price1 + ", $" + price20 );
 }
}
```



Putting it all together

You are now working with THREE files or more e.g.

- The application file e.g. `L14.java`.
- The program class e.g `ProgramClassL14.java`.
- A class definition e.g `Chocolate.java`.

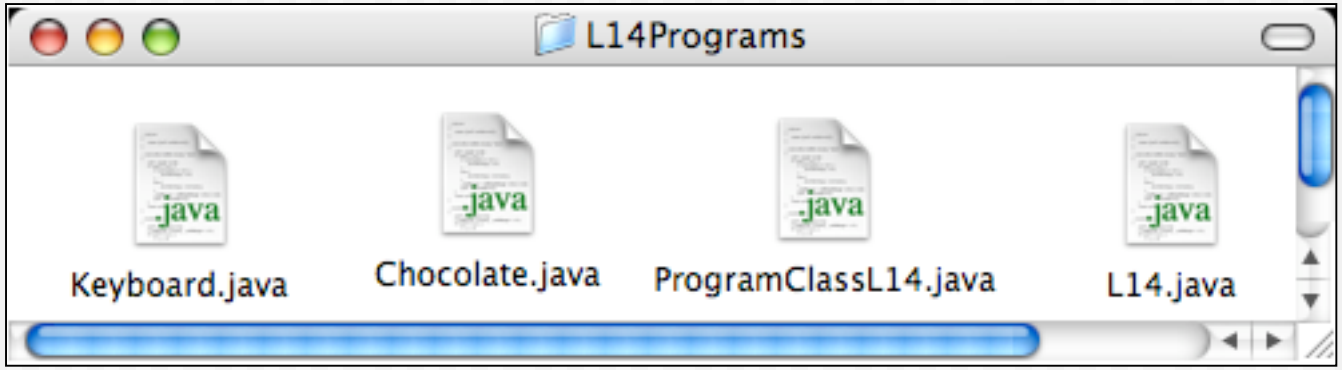
and sometimes, if you are using input from the user:

- The `Keyboard` class i.e. `Keyboard.java`.

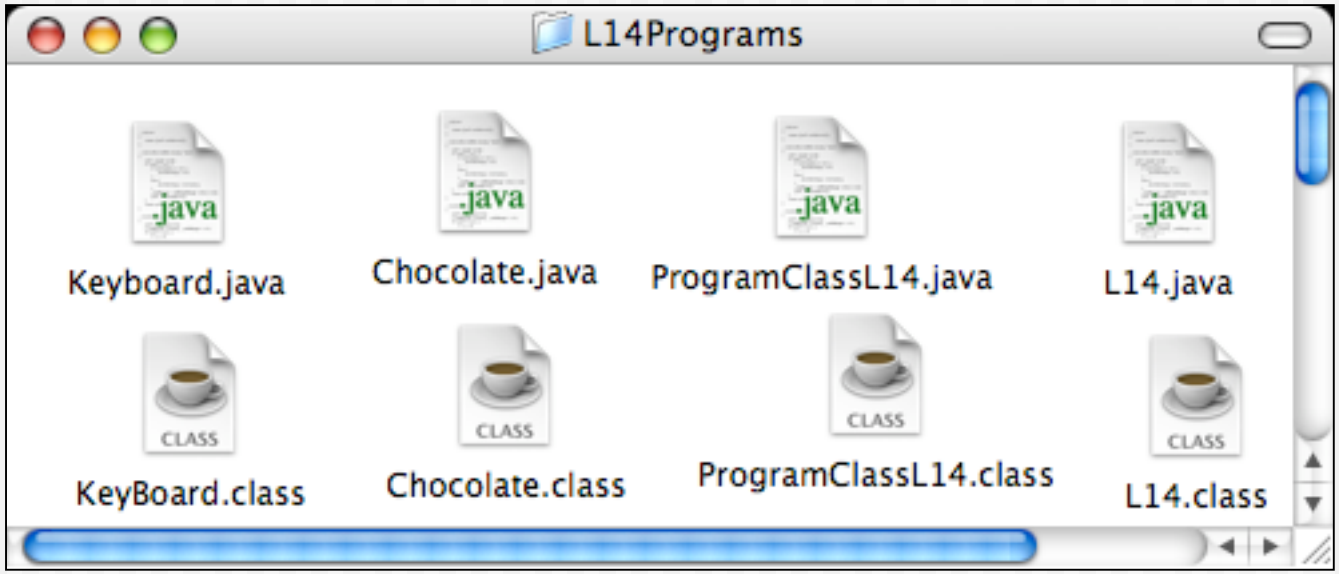
and sometimes even more files, if you are using instances of more than one class.

Putting it all together

The files in your directory



and after they have all been compiled:



Putting it all together

In 101 we define each class in its own file. For example the class definition for the `Chocolate` class is stored in a file called `Chocolate.java`.

After all the classes have been compiled, the application is executed. To run the application:

```
> java L14
....
```

Ex05 - Constructor problem

Using the Chocolate class defined on the previous slides, what is the problem with the following code?

```
public class ProgramClassL14 {
 public void start() {
    Chocolate choc5;
    choc5 = new Chocolate( 333, "Nutaholic" );
 }
}
```

Ex06-instance method problem

Using the Chocolate class defined on the previous slides, what is the problem with the following code?

```
public class ProgramClassL14 {
 public void start() {
   Chocolate choc6;
   choc6 = new Chocolate(334, "Creamy", 1 );
   String desc = choc6.setDescription("MMM!");
 }
}
```

Ex07-instance method problem

Using the Chocolate class defined on the previous slides, what is the problem with the following code?

```
public class ProgramClassL14 {
 public void start() {
    Chocolate choc7;
    choc7 =new Chocolate( 337,"ChocOoze", 1 );
    double price20 = choc7.getPrice20( 1.5 );
 }
}
```

What you need to know

- Objects have state (instance variables) and behaviour (instance methods)
- Class vs objects

- Structure of a class definition
  - Instance variables
  - Constructors
  - Instance methods

- How to create instances of a class
- How to draw a diagram of an instance

```
public class Chocolate {
   private int code;
   private String description;
   private double price;

   public Chocolate(int c,String desc,double p){
      code = c;
      description = desc;
      price = p;
   }

   public double getPrice() {
       return price;
   }
   public double getPrice20() {
       return price * 20 * 0.8;
   }
   public void setDescription(String desc) {
       description = desc;
   }
}
```