2020 CI401
Introduction to programming

# Week 1.07
# Objects and methods

Dr Roger Evans
Module leader

17th November 2020

# Lecture recording and attendance

- This lecture will be recorded and published in the module area
- The focus of recording is on the lecturer, not the audience
- If you are particularly concerned not to be part of the recording, turn off your microphone and camera.

- In addition, lecture attendance is now being routinely recorded (in all modules) to help the School Office monitor engagement

- (This slide is really a reminder to me to start recording and record attendance!)

# Code clinic

- If you need help with programming or databases then our Code Clinic sessions may be the answer. Every weekday in semester 1 we are running 1 hour sessions to help Level 4 and Level 5 students with their programming and databases work.
- The sessions are run by final year computing students and take place on Microsoft Teams. The join code is o727xhp
- Sessions:
  - Mon-Wed, 1pm-2pm
  - Thu, 12pm-1pm
  - Fri, 10am-11am

# Module structure (version 3)

## Semester 1

| Week | Topic | Theme |
|---|---|---|
| 1.01 | Introduction / Hello World | Coding |
| 1.02 | Variables, loops and choices | Coding |
| 1.03 | Input, more loops and choices | Coding |
| 1.04 | Variables and expressions | Coding |
| 1.05 | Types, assignment and arrays | Data |
| 1.06 | Let's play Top Trumps! | Data |
| 1.07 | Objects and methods | OO |
| 1.08 | Working with numbers | Data |
| 1.09 | Simple Algorithms | Dvp |
| 1.10 | Introduction to JavaFX | Dvp |
| 1.11 | Simple Animation | Dvp |
| | Xmas vacation 21 Dec – 8 Jan | |
| 1.12 | GUIs using MVC | OO |
| 1.13 | | |

## Semester 2

| Week | Topic | Theme | Project |
|---|---|---|---|
| 2.01 | Project topics and assessment | Project | Set |
| 2.02 | Simple Inheritance | OO | Lab |
| 2.03 | Scope, Visibility and Encapsulation | OO | Lab |
| 2.04 | Testing - JUnit | Testing | Lab |
| 2.05 | Documentation - Javadoc | Doc | Study |
| 2.06 | Collections and generic types | Data | Study |
| 2.07 | IO: files and streams | Dvp | Study |
| | Easter Vacation 29 Mar – 16 Apr | | |
| 2.08 | Numbers – the computer's view | Data | Study |
| 2.09 | Java vs Python | | Submit? |
| 2.10 | More algorithms – search and sort | Dvp | |
| 2.11 | How fast is my code? | Dvp | |
| 2.12 | Java 'under the hood' | | |
| 2.13 | Revision week | | Exam ↓ |

# Object oriented programming

# Getting organised

- We know how to write simple code to ‘do things’ with simple pieces of data (print things, test things, use arrays and loops etc.)
- Last week we saw how to organise more complex data, by using classes which are templates for creating objects
- This week we learn how to organise our code, and how to combine the organisation of data and code – the basic principle of object-oriented programming

- First, what’s the problem?

# Printing Student info

This class from last week creates a Student object, initialises it, and then prints info about it

To do this it has to know a lot about the structure of a Student object (the names of its variables etc.)

What if we decided to change a Student object? (For example, adding more data or changing a variable name)

We would have to find all the programs where we used it and change them as well.

```java
// creating a Student object and setting values
public class Example2
{
    public static void main(String[] args) {
        Student student1 = new Student();

        student1.firstName = "John";
        student1.surname = "Fisher";
        student1.gender = "Male";
        student1.age = 20;

        System.out.println(student1.firstName);
        System.out.println(student1.surname);
        System.out.println(student1.gender);
        System.out.println(student1.age);
        System.out.println(student1.course);
    }
}
```

# Learning to code WELL

- The problem is that our code is in one place (Example2), and the data is in another (Student).
- One of the key principles of Java, and object-oriented programming in general, is that we should not do this.
- Instead, we should keep data and code that works with that data together, in one place
- So code relating to students, should be in the Student class and code relating to bank accounts should be in the BankAccount class
- This makes it much easier to understand, test and maintain (change) code, especially in large programs.

# Organising our code

- The key to this is to create different blocks of code in different classes, and give each block a name which we can use in other instructions to run the block whenever we want to.
- These named blocks of code are called different things in different programming languages, such as procedures, functions or subroutines.
- In Java we call them methods.

# Methods

# Methods in Java

- Methods are named blocks of code that are associated with classes and objects
- They are declared in a class, like instance variables are.
- Nothing happens when they are declared (the code does not run), they just become available to be used elsewhere
- To run the instructions in a method, you have to call (or invoke) the method (using its name)
- When a method is called from one place in the code, the computer jumps to the method code and runs it, and when it has finished, it jumps back to where it was called from and carries on
- When a method is called, you can pass information to it, called arguments, and it can return a value, called the return value.

# Student2 – adding a method

Class Student2 is just like Student (from last week), but with a method added

The method is called info and it has instructions to print out all the values of the variables

(You can think of it as printing the Top Trumps card for a particular student)

Student2 ×

Compile | Undo | Cut | Copy | Paste | Find... | Close

```java
// Version of the Student class with an info method
public class Student2
{
    String firstName;
    String surname;
    String gender;
    int age = 18;
    String course = "CS";


    // A method to print information about a Student
    public void info() {
        System.out.println(firstName);
        System.out.println(surname);
        System.out.println(gender);
        System.out.println(age);
        System.out.println(course);
    }
}
```

# Student2 – info in detail

The first line declares the method:

- Ignore public and void for now
- info is the name of the method
- The round brackets () tell us this is a method declaration (not a variable, or an array etc.).

The code between { … } is the body of the method

```
    // A method to print information about a Student
    public void info() {
        System.out.println(firstName);
        System.out.println(surname);
        System.out.println(gender);
        System.out.println(age);
        System.out.println(course);
    }
}
```

# Student2 – info more details

When you write code inside a class, the code in the method can use the instance variables of the class directly – you can just say firstName, not student1.firstName (etc)

When the method runs, you give it an object (of the same class – Student2) to run on, which provides the particular values to use

So each Student2 object will print out its own particular values

```
    // A method to print information about a Student
    public void info() {
        System.out.println(firstName);
        System.out.println(surname);
        System.out.println(gender);
        System.out.println(age);
        System.out.println(course);
    }
}
```

# Student2Example

Student2Example is some simple code to use our new method

It is a separate class, which creates a Student2 object and then calls the info method on that particular object

Notice

- The dot notation to call a method on an object
- The round brackets which tell us we are calling a method

Student2Example ×

Compile | Undo | Cut | Copy | Paste | Find... | Close

```java
// creating and printing a Student2 object using the info method
public class Student2Example
{
    public static void main(String[] args) {
        // notice we have to use Student2 as the class and type name, not Student
        Student2 student = new Student2();

        student.firstName = "John";
        student.surname = "Fisher";
        student.gender = "male";

        // call the info method from the Student2 class
        // notice that this means that THIS class doesn't need to know the details
        // of what is printed - it just needs to know there is an info() method.
        // This means you can change what is printed by info inside Student2 and
        // this class will still work.
        student.info();
    }

}
```

# More methods – BankAccount2

- More generally, we can add methods to achieve the main operations associated with a particular class
- For example, BankAccount objects require actions like withdraw and deposit money, but Student objects don't

- BankAccount2 is a version of BankAccount which includes methods for those two actions

BankAccount2 - week1.04

Class Edit Tools Options

BankAccount2 Example3

Compile Undo Cut Copy Paste Find... Close Source Code

```java
// BankAccount class with 2 simple methods
public class BankAccount2
{
    String firstName;
    String surname;
    double balance = 0.00;
    double overdraftLimit = 0.00;

    // Method to deposit money into account
    public void deposit(double amount) {
        balance = balance + amount;
    }

    // Method to withdraw money from account unless overdrawn
    public boolean withdraw(double amount) {
        if (balance - amount >= -overdraftLimit) {
            balance = balance - amount;
            return true;
        } else {
            return false;
        }
    }
}
```

Class compiled - no syntax errors

saved

BankAccount2 - week1.04

Class Edit Tools Options

BankAccount2 × Example3 ×

Compile Undo Cut Copy Paste Find... Close

```
// BankAccount class with 2 simple methods
public class BankAccount2
{
    String firstName;
    String surname;
    double balance = 0.00;
    double overdraftLimit = 0.00;

    // Method to deposit money into account
    public void deposit(double amount) {
        balance = balance + amount;
    }

    // Method to withdraw money from account unless overdraw
    public boolean withdraw(double amount) {
        if (balance - amount >= -overdraftLimit) {
            balance = balance - amount;
            return true;
        } else {
            return false;
        }
    }
}
```

Class compiled - no syntax errors

saved

The first method is called deposit. Ignoring 'public' and 'void' for now, the thing that tells us this is a method is the presence of round brackets ( and ). These are used to wrap around arguments. Each argument is a variable (with type declaration) – here it is the variable amount of type double.

When the method is called, the calling code provides a value between the round brackets (using an expression), and that value is put into this variable.

deposit is a simple method, it just adds amount to balance (one of the instance variables of the class)

BankAccount2 - week1.04

Class Edit Tools Options

BankAccount2 Example3

Compile Undo Cut Copy Paste Find... Close Source Code

```java
// BankAccount class with 2 simple methods
public class BankAccount2
{
    String firstName;
    String surname;
    double balance = 0.00;
    double overdraftLimit = 0.00;

    // Method to deposit money into account
    public void deposit(double amount) {
        balance = balance + amount;
    }

    // Method to withdraw money from account unless overdrawn
    public boolean withdraw(double amount) {
        if (balance - amount >= -overdraftLimit) {
            balance = balance - amount;
            return true;
        } else {
            return false;
        }
    }
}
```

Class compiled - no syntax errors

saved

This is the first line of the withdraw method. The type (boolean) before the method name is called the return type, and tells Java that the method returns a value of this type.

This means the method call can be used in an expression.

deposit has the return type void, which means it doesn't return a value at all. It can't be used in an expression (like println)

BankAccount2 - week1.04

Class Edit Tools Options

BankAccount2 × Example3 ×

Compile Undo Cut Copy Paste Find... Close

```java
// BankAccount class with 2 simple methods
public class BankAccount2
{
    String firstName;
    String surname;
    double balance = 0.00;
    double overdraftLimit = 0.00;

    // Method to deposit money into account
    public void deposit(double amount) {
        balance = balance + amount;
    }

    // Method to withdraw money from account unless overdrawn
    public boolean withdraw(double amount) {
        if (balance - amount >= -overdraftLimit) {
            balance = balance - amount;
            return true;
        } else {
            return false;
        }
    }
}
```

Class compiled - no syntax errors

When the withdraw method is called, amount will be set to a particular value – the amount of money to be withdrawn.

withdraw checks that we are not going over the overdraftLimit (an instance variable), we make the withdrawal (simply by subtracting amount from the instance variable balance) and return true, to say the withdrawal was successful.

Otherwise we return false to say it failed (and don't change the balance).

return is a bit like break – it means 'stop running the method now and go back (return) to wherever we were called from'

BankExample - week1.06-lab

Class Edit Tools Options

BankExample

Compile Undo Cut Copy Paste Find... Close

```
public class BankExample

{
    public static void main(String[] args) {
        BankAccount2 account = new BankAccount2();

        account.firstName = "John";
        account.surname = "Fisher";
        account.overdraftLimit = 100.00;

        System.out.println(account.firstName);
        System.out.println(account.surname);
        System.out.println(account.balance);
        System.out.println(account.overdraftLimit);

        // run the deposit method in BankAccount2 on this account object
        account.deposit(20.00);
        System.out.println("Deposited £20 - balance now: "+account.balance);
        // Run the withdraw method in BankAccount2
        // Because we run it in a while statement, the loop will keep running
        // (and we keep withdrawing money) until the withdraw method returns fals
        // (when we exceed our overdaft)
        while (account.withdraw(40.00)) {
            System.out.println("Another £40 withdrawn: " + account.balance);
        }
        System.out.println("Overdraft limit exceeded");
    }
}
```

Class compiled - no syntax errors

saved

First we create a BankAccount2 object, initialise it to have £100 overdraft limit, and print out all its data.

Then we make our first method call. We call the deposit method for account. Since account is type BankAccount2, the deposit method in that class is used. It takes one argument, which we set to 20.00, and that gets put into the amount variable. The method code runs, and then we come back here to carry on with this code. deposit doesn't return a value so this is just a command.

BankExample - week1.06-lab

Class Edit Tools Options

BankExample

Compile Undo Cut Copy Paste Find... Close

```java
public class BankExample

{
    public static void main(String[] args) {
        BankAccount2 account = new BankAccount2();

        account.firstName = "John";
        account.surname = "Fisher";
        account.overdraftLimit = 100.00;

        System.out.println(account.firstName);
        System.out.println(account.surname);
        System.out.println(account.balance);
        System.out.println(account.overdraftLimit);

        // run the deposit method in BankAccount2 on this account object
        account.deposit(20.00);
        System.out.println("Deposited £20 - balance now: "+account.balance);
        // Run the withdraw method in BankAccount2
        // Because we run it in a while statement, the loop will keep running
        // (and we keep withdrawing money) until the withdraw method returns false
        // (when we exceed our overdaft)
        while (account.withdraw(40.00)) {
            System.out.println("Another £40 withdrawn: " + account.balance);
        }
        System.out.println("Overdraft limit exceeded");
    }
}
```

Class compiled - no syntax errors

saved

The withdraw method is called as the test of a while loop. withdraw returns a boolean value, so it is fine to use it as a test.

When withdraw is called, 40.00 is given as the value of amount. If there's enough money in the account to withdraw £40, it returns true and we run the loop body, which prints a message (including the current balance). Then it goes back and tries to withdraw another £40.

When we reach the overdraft limit, withdraw returns false and the loop stops.

BankExample - week1.06-lab

Class Edit Tools Options

BankExample

Compile Undo Cut Copy Paste Find... Close

```java
public class BankExample

{
    public static void main(String[] args) {
        BankAccount2 account = new BankAccount2();

        account.firstName = "John";
        account.surname = "Fisher";
        account.overdraftLimit = 100.00;

        System.out.println(account.firstName);
        System.out.println(account.surname);
        System.out.println(account.balance);
        System.out.println(account.overdraftLimit);

        // run the deposit method in BankAccount2 on this account object
        account.deposit(20.00);
        System.out.println("Deposited £20 - balance now: "+account.balance);
        // Run the withdraw method in BankAccount2
        // Because we run it in a while statement, the loop will keep running
        // (and we keep withdrawing money) until the withdraw method returns false
        // (when we exceed our overdaft)
        while (account.withdraw(40.00)) {
            System.out.println("Another £40 withdrawn: " + account.balance);
        }
        System.out.println("Overdraft limit exceeded");
    }
}
```

Class compiled - no syntax errors

saved

BlueJ: Terminal Window - week1.04

Options

```
Example3.main({ });
John
Fisher
0.0
100.0
Deposited £20 - balance now: 20.0
Another £40 withdrawn: -20.0
Another £40 withdrawn: -60.0
Another £40 withdrawn: -100.0
Overdraft limit exceeded
```

Can only enter input while your progr

# State and behaviour

BankAccount2 illustrates the typical structure of a Java class, with two main components:

- Instance variables, storing information about each individual object (which may change) – we call this the object's state
- Methods, proving actions that cam be applied to objects – we call this the object's behaviour

Grouping state and behaviour together in classes is a fundamental principle of object-oriented programming

BankAccount2 - week1.04

Class Edit Tools Options

BankAccount2 × Example3 ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
// BankAccount class with 2 simple methods
public class BankAccount2
{
    String firstName;
    String surname;
    double balance = 0.00;
    double overdraftLimit = 0.00;

    // Method to deposit money into account
    public void deposit(double amount) {
        balance = balance + amount;
    }

    // Method to withdraw money from account unless overdrawn
    public boolean withdraw(double amount) {
        if (balance - amount >= -overdraftLimit) {
            balance = balance - amount;
            return true;
        } else {
            return false;
        }
    }
}
```

Class compiled - no syntax errors

saved

# Constructor methods

Student2Example ×

Compile | Undo | Cut | Copy | Paste | Find... | Close

```
// creating and printing a Student2 object using the info method
public class Student2Example
{
    public static void main(String[] args) {
        // notice we have to use Student2 as the class and type name, not Stu
        Student2 student = new Student2();

        student.firstName = "John";
        student.surname = "Fisher";
        student.gender = "male";

        // call the info method from the Student2 class
        // notice that this means that THIS class doesn't need to know the de
        // of what is printed - it just needs to know there is an info() meth
        // This means you can change what is printed by info inside Student2
        // this class will still work.
        student.info();
    }

}
```

Here is Student2Example again. In the first line of the main code we create an object with new Student2(). This object has default values for all its instance variables.

In the next three lines we assign more specific values to some of these variables.

Constructor methods allow us to do both steps in one go – create and initialise an object.

This is neater, more flexible, and means the initialisation code is in the class file, not in the main program.

# Constructor methods

- Constructor methods are just like other methods, except their header line is slightly different. In the Student class, a constructor method would start like this:

```
public Student() { … }
```

- You can identify a constructor method because its name is always the name of the class, and it has no return type at all!
- Constructor methods appear after the new keyword to create new objects, as in:

```
Student student1 = new Student();
```

Student3 ×

Compile | Undo | Cut | Copy | Paste | Find... | Close

```java
public class Student3
{
    String firstName;
    String surname;
    String gender;
    int age = 18;
    String course = "CS";

    // The default constructor method which does nothing - so the values are as
    // specified in the variable declarations
    public Student3() {
    }

    // A constructor method which allows you to specify name and gender
    public Student3(String myFirstName, String mySurname, String myGender) {
        firstName = myFirstName;
        surname = mySurname;
        gender = myGender;
    }

    // A constructor method which allows you to specify everything
    public Student3(String myFirstName, String mySurname, String myGender, int myAge, String myCourse) {
        firstName = myFirstName;
        surname = mySurname;
        gender = myGender;
        age = myAge;
        course = myCourse;
    }
```

Here is the Student3 class which has three constructor methods.

The first one just does nothing – which is what our calls to new Student3() do.

The second method lets you specify the firstName, surname and gender as arguments, and then assigns them to the instance variables. (The other two variables just get their default values)

The third method allows us to specify everything to make a Student3 object exactly as we wish.

# Using constructor methods

- Constructor methods appear after the new keyword:

```
Student3 s1 = new Student3();
```

- The ‘empty’ constructor method (with no arguments, and no code) is created automatically by Java if you don’t provide one. So you can always use it with a class.

- But if you do provide methods, you can use those too

```
Student3 s2 = new Student("John", "Fisher", "male");

Student3 s3 =
    new Student("Mary", "Jones", "female", 24, "CSAI");
```

# Note: method overloading

- Notice in Student3 we have three methods with the same name
- This is called method overloading
- Java decides which method to call depending on how many arguments you provide (and what types they are) – we call this information the method's signature
- You can use method overloading with the other method types too, not just constructor methods

# Constructors and arrays of objects

- In last week's lab we used a class in an array type, to make an array of Student objects.
- This looked something like this:
  - ```
    Student student1 = new Student();
    Student student2 = new Student();
    Student student3 = new Student();

    Student[] classList = {student1, student2, student3};
    ```
- Our new constructor methods make this even easier
    ```

# Constructors and arrays – Student3Example

With our new constructor methods we can create different individual students without needing the variables student1, student2 and student3 at all.

And we can make a for loop to process them using the info method.

Notice how this version of the program does not need to know anything about the internal structure of the Student3 class

```java
// creating and printing Student3 objects using constructors and the info method
public class Student3Example
{
    public static void main(String[] args) {
        // notice we have to use Student3 as the class and type name, not Student,
        // and we can mix up constructor methods if we want to
        Student3[] classList = {
            new Student3("John", "Fisher", "male"),
            new Student3("Mary", "Jones", "female"),
            new Student3("Kevin", "James", "male", 25, "CSAI")
        };

        // call the info method from the Student3 class
        // notice that this means that THIS class doesn't need to know the details
        // of what is printed - it just needs to know there is an info() method.
        // This means you can change what is printed by info inside Student3 and
        // this class will still work.
        for (Student3 s: classList) {
            s.info();
        }

    }
}
```

# Methods summary

# Three kinds of method

- We have talked about three different kinds of method today:

```
1. public boolean withdraw(double amount) { … }
2. public void deposit(double amount) { … }
3. public Student3(…) { … }
```

- In (1) the type specifier boolean (the return type) tells us that withdraw returns a value and so can be used as part of an expression (you can use any type – boolean is especially useful for method tests).
- In (2) the special type specifier void tells us that deposit does not return a value, so it can not be used as part of an expression, but only as a statement. (info in Student3 was also of this kind)
- In (3) there is no return type at all and the name is the same as the class – this is a constructor method, used after the new keyword

# Another method type

- As we saw last week, we have actually been using another method type ever since week 1.01
- HelloWorld (and every other lab program) has a method that started:

```
4. public static void main(String[] args) { … }
```

- This exactly like type (2), with the addition of the word static, and its name is (nearly) always main – it is called the main method
- When you run a Java class as a program, if you don't tell it to do something else, it looks for a static method called main and runs that.
- So this is how you get your program to start up.

# Linking between objects

# Linking between objects

- As well as making arrays of objects of different sorts another important idea is that we can link objects of different sorts together.
- Going back to our card analogy, we have mostly thought of the things written on the card as primitive types – numbers, booleans and strings (which are not really primitive, but we often treat them that way).
- But we can use any type for a piece of information on our card, including another class, which amounts to saying that we are making a link from one card to another.

# Linking between objects

- Here are some of our cards for bank accounts and students
- Notice how the name information is duplicated between the two
- That is not great
  - inefficient use of memory
  - difficult to maintain
  - annoying for the user

| | |
|---|---|
| firstName | David |
| surname | Smith |
| balance | £20 |
| overdraftLimit | £100 |

| | |
|---|---|
| firstName | David |
| surname | Smith |
| gender | male |
| age | 19 |
| course | CS |

| | |
|---|---|
| firstName | Mark |
| surname | Jones |
| balance | £500 |
| overdraftLimit | £100 |

| | |
|---|---|
| firstName | Mark |
| surname | Jones |
| gender | male |
| age | 18 |
| course | CSG |

# Linking between objects

- Here's the template for the BankAccount objects
- Suppose we replace the firstName and surname slots with a single one, say accountHolder of type Student3
- Then when we create a BankAccount object, we have to provide a Student3 object for this slot

| | |
|---|---|
| String firstName | |
| String surname | |
| double balance | 0.00 |
| double overdraftLimit | 0.00 |

| | |
|---|---|
| Student3 accountHolder | |
| double balance | 0.00 |
| double overdraftLimit | 0.00 |

# Linking between objects

The code for that might look like this:

```
Student3 student1 =
   new Student("Mark", "Jones", "male");

BankAccount account = new BankAccount();

account.accountHolder = student1;
```

| | |
|---|---|
| String firstName | |
| String surname | |
| double balance | 0.00 |
| double overdraftLimit | 0.00 |

| | |
|---|---|
| Student3 accountHolder | |
| double balance | 0.00 |
| double overdraftLimit | 0.00 |

# Linking between objects

- In pictures this might look like this.
- In terms of cards, the best way to think of it is as a line drawn, or a piece of string linking, from the value on one card, to a whole other card (in a different deck)
- The benefit is you only need to change a name etc in one place, not two.

| | |
|---|---|
| accountHolder | |
| balance | £20 |
| overdraftLimit | £100 |

| | |
|---|---|
| firstName | David |
| surname | Smith |
| gender | male |
| age | 19 |
| course | CS |

| | |
|---|---|
| accountHolder | |
| balance | £500 |
| overdraftLimit | £100 |

| | |
|---|---|
| firstName | Mark |
| surname | Jones |
| gender | male |
| age | 18 |
| course | CSG |

# Linking between objects

How do we now find out our account holder's surname?

```
Student3 student1 = new
Student3("Mark", "Jones", "male");

BankAccount account =
   new BankAccount();

account.accountHolder = student1;
```

Instead of

```
account.surname
```

we write

```
account.accountHolder.surname
```

| | |
|---|---|
| accountHolder | |
| balance | £20 |
| overdraftLimit | £100 |

| | |
|---|---|
| firstName | David |
| surname | Smith |
| gender | male |
| age | 19 |
| course | CS |

| | |
|---|---|
| accountHolder | |
| balance | £500 |
| overdraftLimit | £100 |

| | |
|---|---|
| firstName | Mark |
| surname | Jones |
| gender | male |
| age | 18 |
| course | CSG |

# Summary

# Object oriented programming

- Methods are named pieces of code which allow us to break our programs into smaller parts
- In object oriented programming, we group data (instance variables) and actions (methods) together in classes. We sometimes call these the state and behaviour of objects in the class
- This makes it easier to write bigger programs without getting confused – Students and Bank Accounts state and behaviour are kept separate

# Kinds of method

- 4 kinds of method:

```
1. public boolean withdraw(double amount) { … }
2. public void deposit(double amount) { … }
3. public Student3(…) { … }
4. public static void main(String[] args) { … }
```

- (1) methods with a return type can be used in expression
- (2) methods with return type void are commands (and can't be used in an expression)
- (3) methods with no return type at all and named the same as the class are constructor methods, used after the new keyword
- (4) methods with the word static, and usually called main are used to start a program running

# Advanced uses of objects

- When you make a new class, you can use it as a type
- For example, with a Student class this means you can:
  - Declare a variable of type Student
  - Create an array of students, with type Student[]
  - Use Student as the type of an instance variable in another class (linking from one ‘card’ to another)
  - Use Student as an argument or return type of a method

# Lab exercises Week 1.07

# Lab exercises – coding

- The Student classes are mainly to support the lecture – no specific labs for them
- The Labs for the BankAccount classes are the main labs for this week
- The Chick classes and Labs are there to provide you with some additional code to play with if you want to.

# Lab exercises 1.07

- Lab1 – Play with the blueJ data inspector and debugger (see slides below)
- Lab2 – Add constructor methods to BankAccount
- Lab3 – Create a StudentBankAccount class which links to a Student3 object

# Using the BlueJ object inspector, code pad and debugger

# BlueJ debugger tutorial links

- BlueJ tutorial chapters 6 and 7
  https://www.bluej.org/tutorial/tutorial-v4.pdf
- Youtube video from BlueJ team
  https://www.youtube.com/watch?v=ji7Ed65BaPI

# Student examples

- Compile all the lab files
- Right-click on Student3 and select new Student3() (to make an instance) – call it myStudent
- Double click Student3 instance to inspect (making a Top Trumps ‘card’)
- Move the ‘card’ (inspection window) so it doesn’t overlap with Bluej window
- Use the code pad to show field values – eg type
  - myStudent.age;
- Use code pad to set values – observe change in ‘card’ – eg type
  - myStudent.firstName=“Paul”;

# BlueJ debugger - BankExample

- Double click BankExample to display the source
- Click the left hand end of the line in the while loop to create a break point (the whole line will turn red)
- Run the program – it will stop just BEFORE it runs the red line of code
- When it stops the debugger window will appear
- Double click on ‘account’ in Local variables – the red instance viewer will appear showing the state of the BankAccount2 instance
- ‘Step’ the debugger round the loop and notice the balance value changing
- ‘Step into’ to show the transition into the withdraw call in BankAccount2, stepping through that and then back to BankExample.
- Do this a couple of times to show the final path through the withdraw, returning false and exiting the loop.