

# COM6516

# Object Oriented Programming and Software Design

The contents of this module has been developed by Adam Funk, Kirill Bogdanov, Mark Stevenson, Richard Clayton and Heidi Christensen

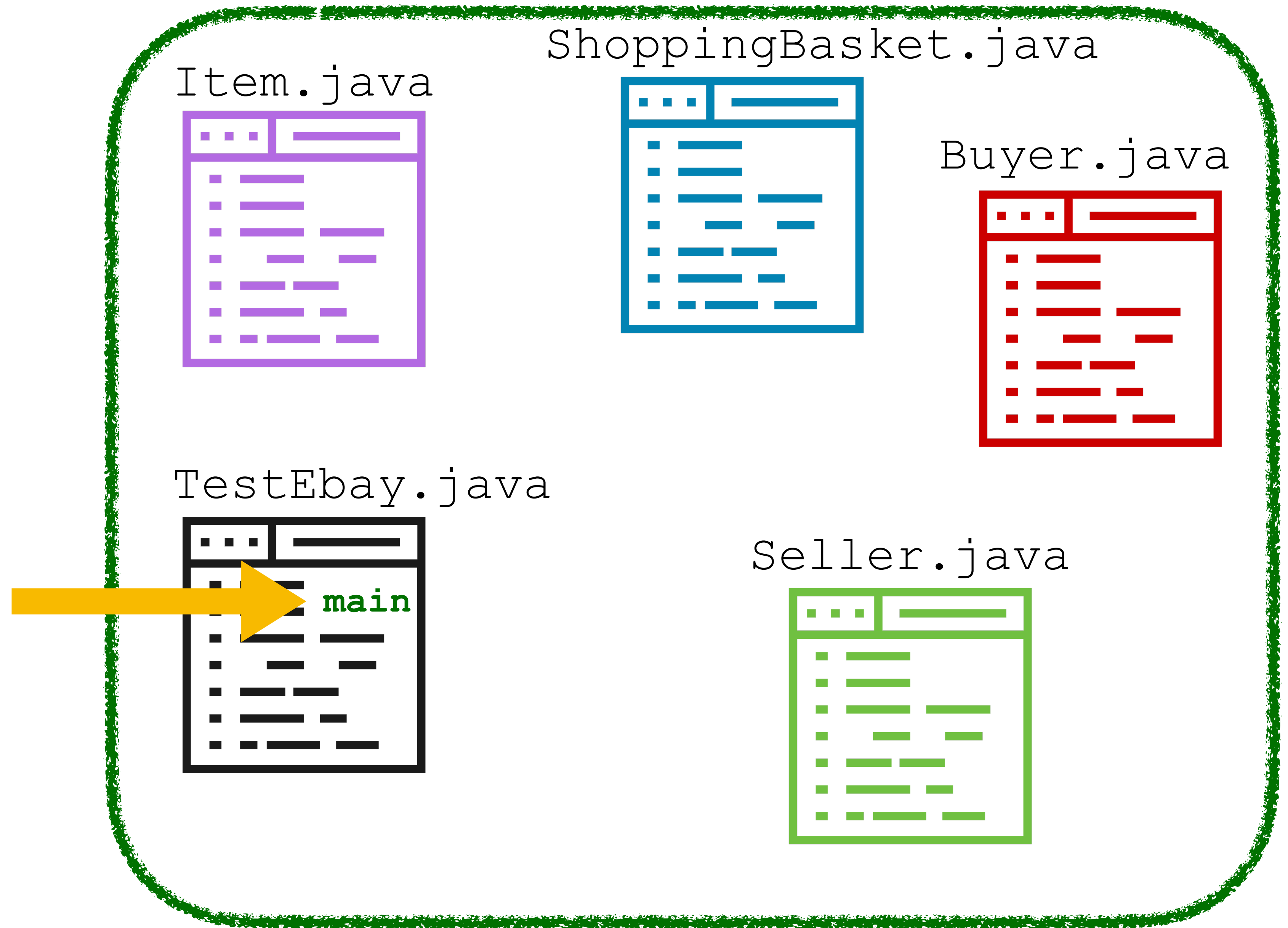
# Practical 2

## Classes and objects

- Classes with a `main` method
- Testing class behaviour
- Instance fields — public or private?
- Class `Complex`
- Using `equals` methods

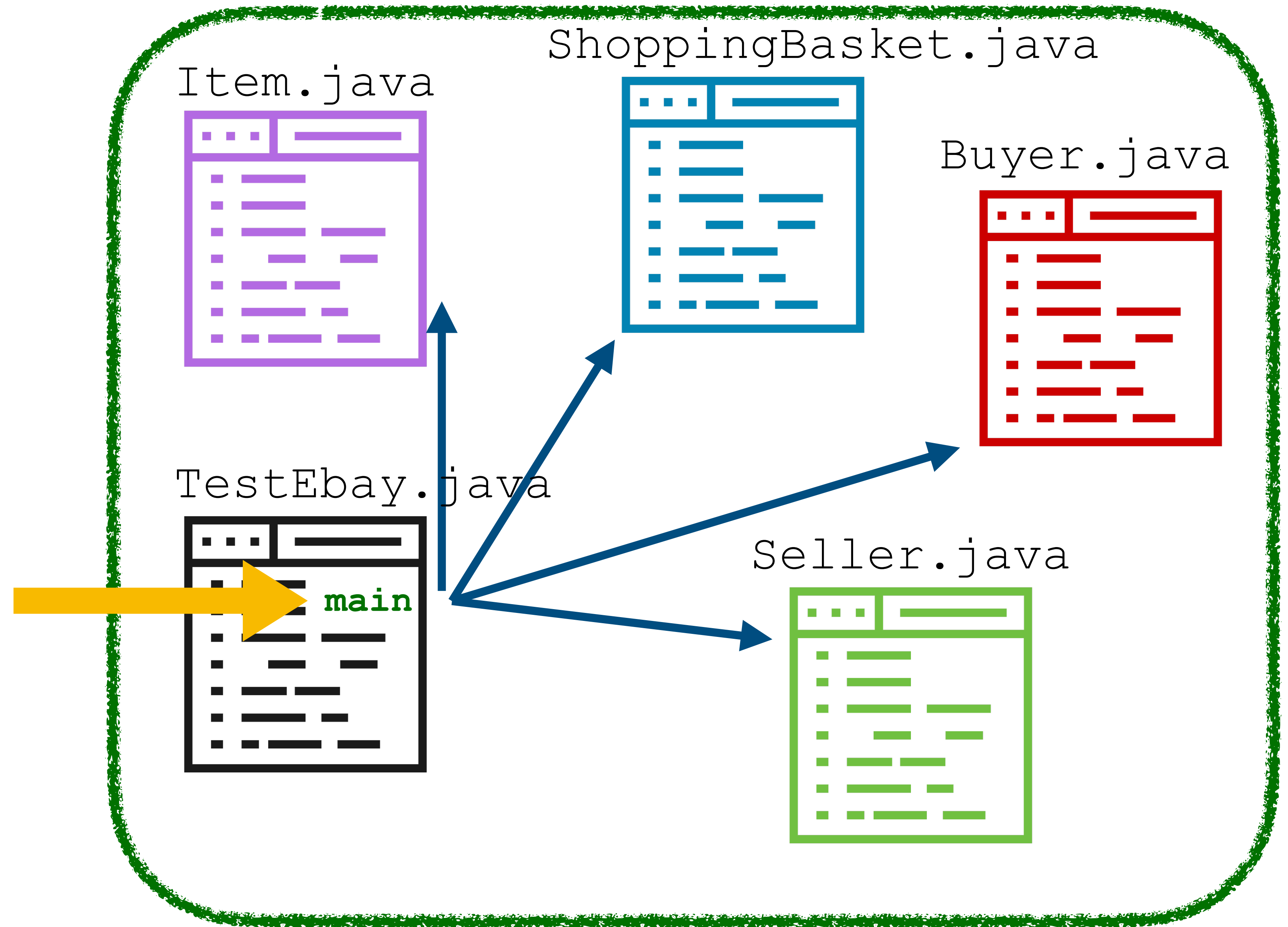
# Classes with a `main` method

You may have several files with classes in your software...



# Classes with a main method

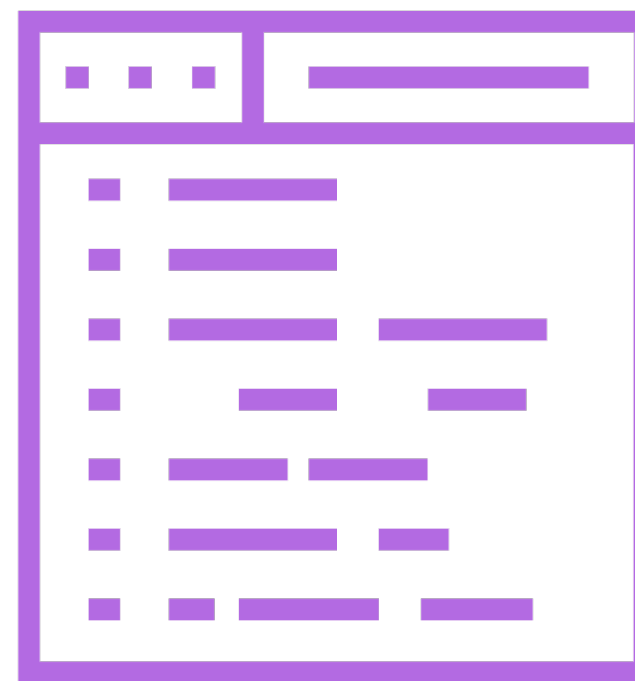
You may have several files with classes in your software...



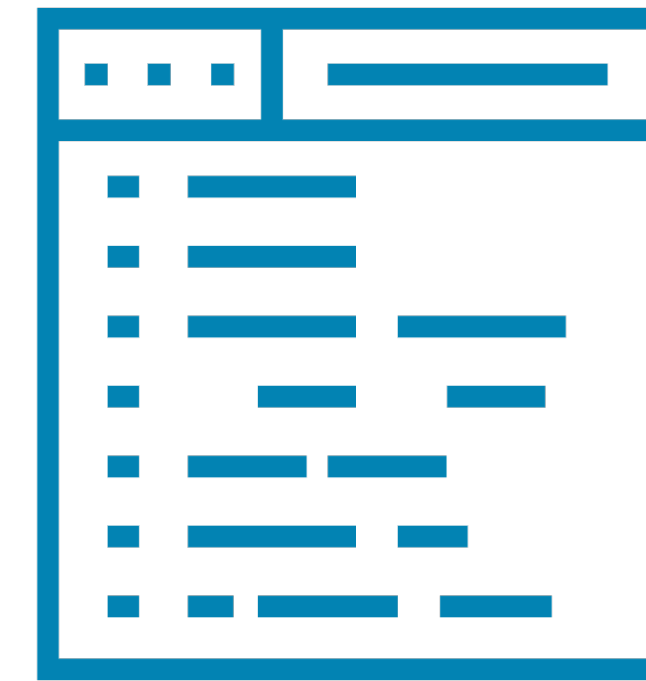
# Classes with a main method

You may have  
several files with  
classes in your  
software...

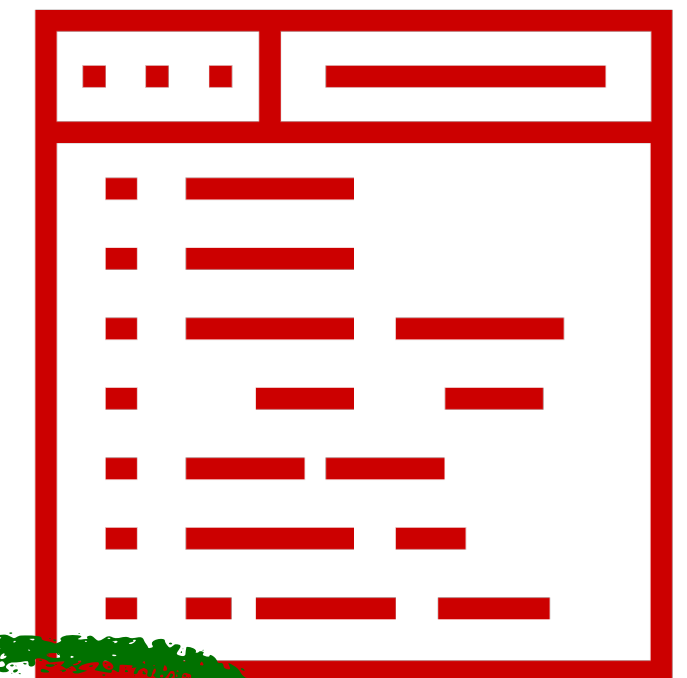
Item.java



ShoppingBasket.java



Buyer.java



TestSeller.java



Seller.java



# Testing class behaviour

- Writing and using test classes helps you to get code working quickly
- The class below has a main method that will be invoked when the bytecode interpreter executes it

```
public class TestFoodStore {  
    public static void main(String[] args) {  
        // create a new FoodStore object called MyFoodStore  
        // by invoking the constructor  
        FoodStore MyFoodStore = new FoodStore(10);  
  
        // display the amount stored by calling the getAmountStored  
        // method associated with the MyFoodStore object  
        System.out.println("Contains " + MyFoodStore.getAmountStored());  
    }  
}
```

# Testing class behaviour

- Writing and using test classes helps you to get code working quickly
- The class below has a main method that will be invoked when the bytecode interpreter executes it

```
/**
 * test class for Customer
 */
public class CustomerTest {
    public static void main(String[] args) {
        // create a new customer
        Customer firstCustomer = new Customer("A.Client", "Sheffield");
        // print out customer information
        System.out.println(firstCustomer.toString());
    }
}
```

# Testing class behaviour

- If we have two files called `CustomerTest.java` and `Customer.java`, then the code can be compiled by either

```
javac CustomerTest1.java
```

or

```
javac Customer*.java
```

- Both approaches will create class files `CustomerTest.class` and `Customer.class`
- Running `CustomerTest` by typing `java CustomerTest` will execute the `main` method



# Instance fields — public or private?

If we had defined our instance field as follows

```
// instance field  
public int numDeposits;
```

we could access the value from code outside the class, and we could change it

```
FoodStore MyFoodStore = new FoodStore(10);  
...  
System.out.println("number of deposits " + MyFoodStore.numDeposits);
```

This breaks encapsulation and is not good practice because the class no longer has control over its instance fields — e.g., it may have other fields that have to be recomputed once this one changes

# Instance fields — public or private?

We make the field private and prepare the accessor method

```
// instance field
Private int numDeposits;
// accessor method
public int getNumDeposits() {
    return(numDeposits);
}
```

and access the value through the method

```
FoodStore MyFoodStore = new FoodStore(10);
...
System.out.println("number of deposits "
    + MyFoodStore.getNumDeposits());
```

# Class Complex

```
public class Complex {  
    private double realPart;  
    private double imagPart;  
    public Complex(double r, double i) {  
        realPart = r;  
        imagPart = i;  
    }  
    public double getReal() {  
        return realPart;  
    }  
    public double getImag() {  
        return imagPart;  
    }  
    public Complex add(Complex c) {  
        return (new Complex(realPart+c.getReal(), imagPart+c.getImag()));  
    }  
}
```

# Class Complex

Easy

Implementing an operator as an instance method:

```
public Complex add(Complex c) {  
    return (new Complex(realPart+c.getReal(), imagPart+c.getImag()));  
}
```

```
Complex sum = c1.add(c2);
```

Easy

Implementing an operator as a class method:

```
public static Complex add(Complex c1, Complex c2) {  
    return (new Complex(c1.getReal()+c2.getReal(),  
                        c1.getImag()+c2.getImag()));  
}
```

```
Complex sum = Complex.add(c1,c2);
```

# Using `equals` methods

## Meaningful comparisons

Testing for `equals` involves distinguishing between **identity** and **equality**

- Two objects that refer to the same **memory location** are **identical**
- Two objects that have the same **state**, or the same **behaviour** are **equal**

To test whether two objects are **equal**, we have to write `equals` methods

# Using equals methods

It generally involves the following steps:

- Test whether the two objects are **identical** (have the same reference)
- Test whether the other object is `null`
- Test whether the objects belong to the same class
- Compare all instance fields, using `==` for primitive types, and `equals` methods for objects

See the following article for an extended discussion —

<http://www.angelikalanger.com/Articles/JavaSolutions/SecretsOfEquals/Equals.html>

# Using equals methods

```
public class ItemWithEquals {  
    ...  
    public boolean equals(Object obj) {  
        if (this == obj) {    // check if identical objects  
            return true;  
        }  
        else if (obj == null) {  
            return false;    // false if parameter is null  
        }  
        else if (this.getClass() != obj.getClass()) {  
            return false;    // false if objects have different classes  
        }  
        else {    // do something specific for Item  
            ItemWithEquals otherItem = (ItemWithEquals) obj;  
            return (name.equals(otherItem.getName()) &&  
                price == otherItem.getPrice());  
        }  
    }  
}
```

**Point to the same memory loc**





# Using equals methods

Test `Item` objects using the `equals` method from the `ItemWithEquals` class:

```
public class TestItemEquals {  
    public static void main(String[] args) {  
        ItemItemEquals itemA = new ItemItemEquals("baked beans", 0.3);  
        ItemItemEquals itemB = new ItemItemEquals("tomato soup", 0.4);  
        ItemItemEquals itemC = new ItemItemEquals("baked beans", 0.3);  
        ItemItemEquals itemD = item1;  
        String testObject = "Hello World";  
  
        System.out.println(itemA.equals(itemB));           // false  
        System.out.println(itemA.equals(itemC));           // true  
        System.out.println(itemA.equals(itemD));           // true  
        System.out.println(itemA.equals(testObject));      // false  
        ...  
    }  
}
```

name and price are all the same