2020 CI401
Introduction to programming

Week 1.04
Variables, expressions and types

Dr Roger Evans
Module leader
27th October 2020

Lecture recording

- 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.

- (This slide is really a reminder to me to turn recording on!)

Remote learning code of conduct

- The university has published a Remote Learning Code of Conduct
- You can find it at https://permalink.brighton.ac.uk/to/codeofconduct , and there is also a link to it on the My Studies module dashboard (under 'Module Resources')
- Quick summary:

Do:

- Be respectful of others as you would face to face
- Join meetings with camera and mic off
- Use 'Chat' carefully – everyone can see it
- Respect confidentiality in chat or discussions
- Choose images (avatars etc) carefully

Don't:

- Do anything that might be considered discriminatory against, or bullying and harassment of, anyone else (students, staff or anyone)

Review of week 1.03

- Input and Output (IO)
  - Human IO
  - Computer IO
- IO in Java
  - System.out (messages on screen)
  - System.in (input from keyboard)
- Scanner – library for getting input
  - next() – get a single word
  - nextInt() – get a (whole) number
  - nextline() – get a whole line of text

- while loops
  - Looping until a test is not true
- switch statements
  - Special kind of if statement for multiple tests of the same kind
- break
  - a special command to end a switch or loop statement ‘in the middle’

- Queen bot
- Estate agent bot (with input)

Introduction to variables

Remembering things in instructions

- When we give or follow instructions we often give things names to help us remember them and combine them
- For example suppose we want to buy paint for a wall
- We need to know:
  - the width and height of the wall so we can work out the area
  - the coverage per litre so we can work out how much paint we need
  - the cost per litre so we can work out the cost
  - oh, and the colour and type of paint we want (nearly forgot that!)

Buying paint

- So we might write down:

width	4m
height	2.5m
area (width x height)	10 sq m
coverage	5 sq m per litre
amount needed (area/coverage)	2 litre
cost per litre	£6.00
total cost (cost per litre x amount needed)	£12.00
colour	light blue
type	eggshell emulsion

Names in programming

- We do exactly the same thing in programming
- We give names to the quantities and other information we need
- And we use those names in calculations
- When we do this, we call the names variables

Buying paint in Java

width	4m
height	2.5m
area (width x height)	10 sq m
coverage	5 sq m per litre
amount needed (area/coverage)	2 litre
cost per litre	£6.00
total cost (cost per litre x amount needed)	£12.00
colour	light blue
type	eggshell emulsion

```
width = 4;
height = 2.5;
area = width * height;
coverage = 5;
amountNeeded = area/coverage;
costPerLitre = 6.00;
totalCost  = costPerLitre * amountNeeded;
colour = "light blue";
type = "eggshell emulsion";
```

Buying paint in Java

- Notice:
  - No ‘units’ (metres, litres etc)
  - Punctuation – = and ;
  - = is not ‘equals’ – it is an assignment statement, and stores the value on the right in the variable on the left
  - Use * for multiplied by
  - Use the names to calculate things for us (eg area = width * height;)
  - How we write namesWithMultipleWords
- width, height, area, amountNeeded etc are variables

```
width = 4;
height = 2.5;
area = width * height;
coverage = 5;
amountNeeded = area/coverage;
costPerLitre = 6.00;
totalCost  = costPerLitre *
amountNeeded;
colour = "light blue";
type = "eggshell emulsion";
```

Variables are more than just names

- Names in ordinary language are often quite fixed things
    - my name belongs to me, and although there may be other people with the same name, we wouldn’t expect my name to belong to someone else tomorrow, and a different person the day after.

- Variables are names for things which can change (they can vary, hence the name)
    - For different runs of a program
    - During the course of running a program

Variables as inputs or parameters

- We use variables to give names to information provided when the program starts, or as input as the program is running
- In the buying paint example, width, height, coverage, costPerLitre, colour and type are examples – they are fixed for one run of the program, but can be different for different runs
- Often we distinguish inputs, which are provided by the user as the program is running, and parameters, which are set when the program starts
- So we might write a program which asks the user for the width, height, colour and type of paint (inputs), but has 'built-in' values for coverage and costPerLitre (parameters)

Variables that change as the program runs

- Variables can also be used to calculate and keep track of things as the program runs
- In the buying paint example, area, amountNeeded and totalCost are variables to hold values which are calculated, so they depend on other variables, and can be used in other calculations
- In our loop examples, there is almost always a variable that changes each time round the loop, and is tested to decide when the loop should end. We call these loop variables.
- For example room in EstateAgent, name in QueenBot

Variables that update themselves

- One special kind of variable usage which we have seen already is a variable which updates its own value
- In the while loop example last week we saw:

```
x = x - 1;
```

- This uses the (old) value of x to calculate another value, and store it as the (new) value of x
- Another common example is for keeping a running total in a loop:

```
runningTotal = runningTotal + cost;
```

- Here we are adding cost to runningTotal each time round a loop, so runningTotal gets bigger and bigger

Choosing variable names

- As a programmer, you get to choose the names of your variables.
- Here are some rules and guidelines about choosing well (in Java):
    - A variable name can contain letters and numbers but it must start with a letter. (It can have $ and _ in it too, but they should normally be avoided.)
    - A variable name should start with a lowercase letter. This distinguishes it from a class name which starts with an upper case letter (Eg String).
    - Variable names should generally be meaningful. Multi-word names should be joined together using camel-case – eg firstName, userPreferenceList.
    - Exceptions are loop variables which are often single letters i, j, m, n, x, y, z etc.
    - Also names for really fixed values (‘constants’) are conventionally all uppercase, with underscores to separate words – eg PI, MAX_SCREEN_WIDTH

Expressions

Expressions

- Expressions are bits of Java code which refer to ‘things’ inside the world of the computer. We call the ‘thing’ the value of the expression.
- Basically there are two kinds of expressions
  - Literal expressions – where you just write the value you want
  - Complex expressions – where you ‘do sums’ with expressions to make new values
- Expressions are instructions to make the program do something, just like statements are – for example the expression 2*3+5 says ‘multiply 2 by 3 and then add 5)
- The special thing about expressions is that they return (‘give you back’) the value they calculate, so the program can do something with it (like print it)

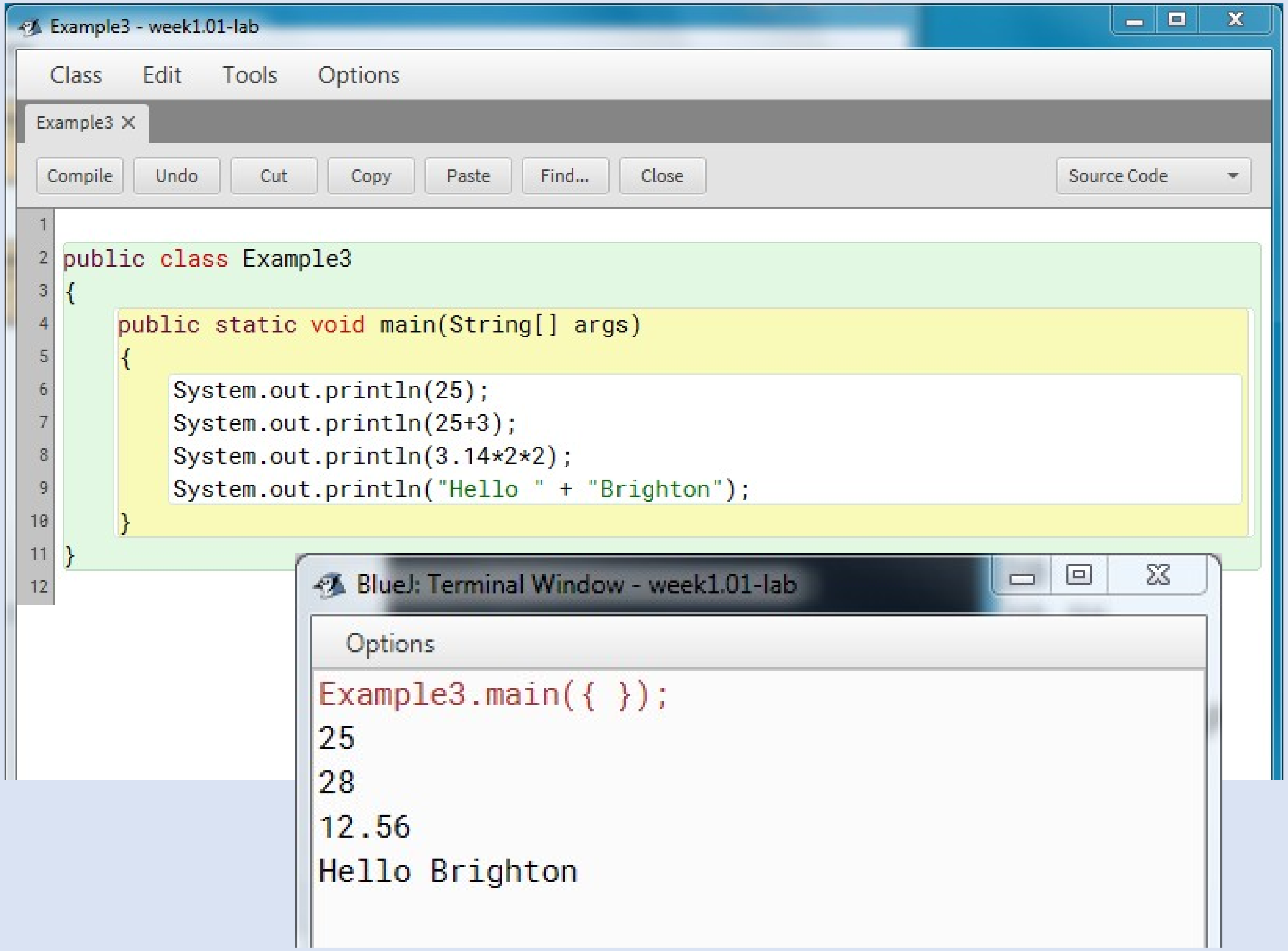
Literal expressions

- The simplest expressions are literal expressions, such as numbers (5, 3.14, 27), strings (“Hello World”), booleans (true and false) and a special one called null (which means ‘nothing’, approximately).
- The value of a literal expression is the expression itself, in internal format in memory. For example, the value of the expression 5 (which is a string containing one character) is ‘the number 5’, which might be represented in the computer in binary as 101, or possibly 00000101 (8 bits), or even 00000000000000000000000000000101 (32 bits)
- (We don’t actually care much what the internal form is, as long as the computer can count and do sums with it correctly)

More interesting expressions

- We can make more interesting expressions (known as complex expressions) by ‘doing sums’ using operators and brackets
- The easiest example is with numbers, where the literal expressions are numbers and the operators are + (add), - (subtract), * (multiply) and / (divide).
- So complex expressions for numbers are things like 5*2, 10-6 etc.
- You can combine complex expressions too, to make more complex expressions, such as 5*2+3, and use brackets, such as (5*2)+(3*4)
- The standard mathematical rules of precedence apply (BIDMAS)

Simple expressions from week 1.01


Example3 - week1.01-lab
Class Edit Tools Options
Example3
Compile Undo Cut Copy Paste Find... Close
Source Code
BlueJ: Terminal Window - week1.01-lab
Options

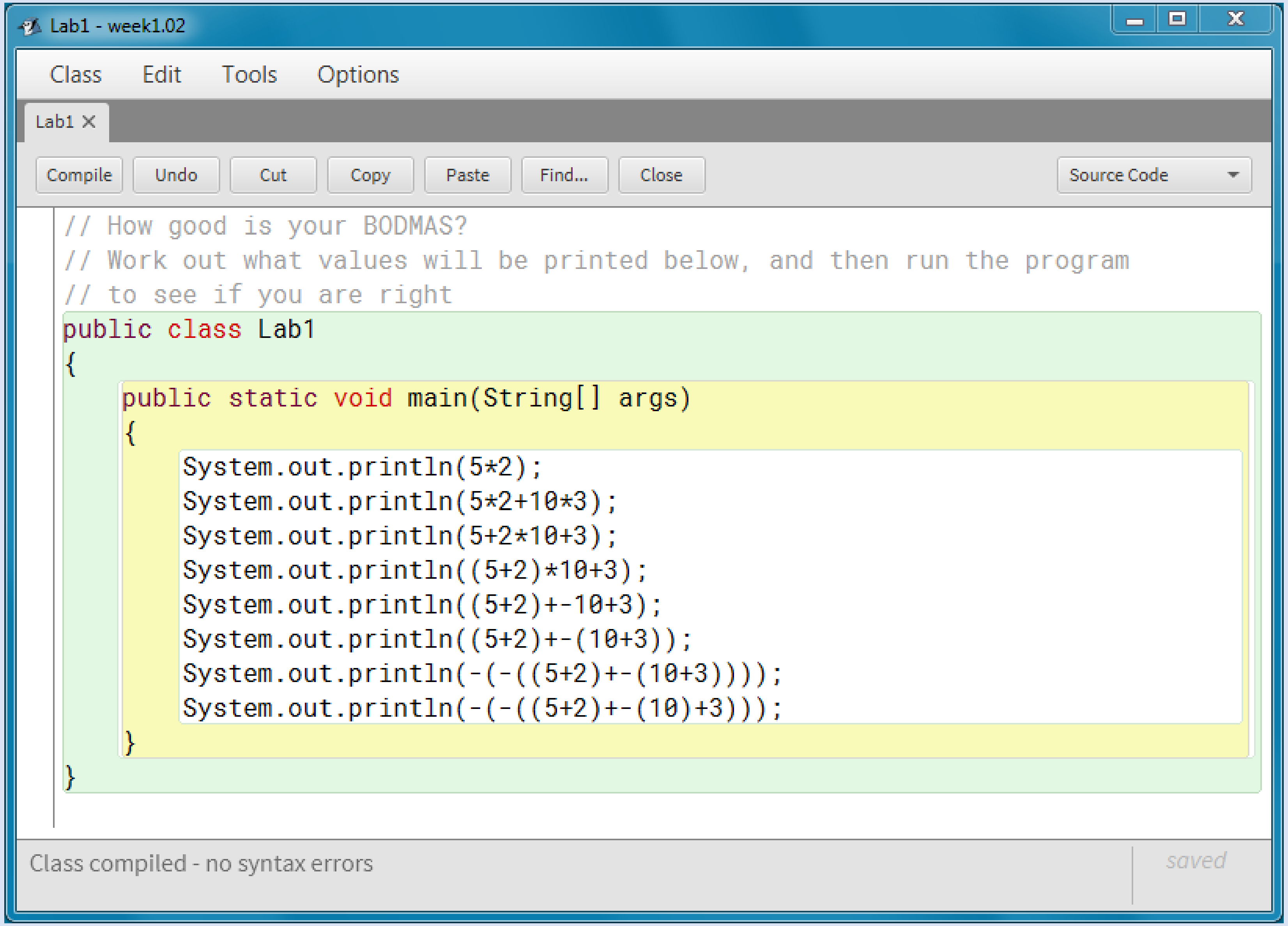
```
public class Example3
{
    public static void main(String[] args)
    {
        System.out.println(25);
        System.out.println(25+3);
        System.out.println(3.14*2*2);
        System.out.println("Hello " + "Brighton");
    }
}
```

```
Example3.main({ });
25
28
12.56
Hello Brighton
```

More expressions

Notice

- (Round) brackets
- Precedence
  * / before + - etc
- Unary –
  -5 -(10+2)



```
// How good is your BODMAS?
// Work out what values will be printed below, and then run the program
// to see if you are right
public class Lab1
{
    public static void main(String[] args)
    {
        System.out.println(5*2);
        System.out.println(5*2+10*3);
        System.out.println(5+2*10+3);
        System.out.println((5+2)*10+3);
        System.out.println((5+2)+-10+3);
        System.out.println((5+2)+-(10+3));
        System.out.println(-(-((5+2)+-(10+3))));
        System.out.println(-(-((5+2)+-(10)+3)));
    }
}
```

Class compiled - no syntax errors

saved

String expressions

- In Java you can make complex expressions out of other kinds of literal, apart from numbers
- In particular you can use the + operator with Strings
  - "Hello" + "World" – combining two strings to make a new string ("HelloWorld" – if you wanted a space you needed to include it 🏝)
  - "Your number is: " + 7
  - "A string with " + 3 + " sections (don't forget the spaces!)"

Boolean expressions (tests)

- We mentioned above that there are two literal expressions called boolean expressions: true and false
- These expressions, and the boolean operators that go with them are used to do logic in a program
- They are named after a famous logician George Boole (1815-1864)
- We have seen them in use already – the tests in if statements and while loops are actually boolean expressions

Understanding boolean expressions

- As you know from if statements, we use tests to make choices, doing one thing if the test is true, and another if it is false
- Computers treat this absolutely literally: a test is a boolean expression which is a set of instructions to calculate a value. This value is either true or false (there are no other boolean values). If the value is true you do one thing, if it is false you do the other
- This is not quite the way we think of choices. We think “if it’s raining I will wear a coat” and look to see if it’s raining. We don’t think “if ‘it’s raining’ is true then I will wear a coat”. But that’s what a computer does.

Boolean values

- There are just two boolean values, true and false
- On their own they are not very exciting for making choices:
  - `if (true) { do X } else { do Y }` will always do X
  - `if (false) { do X } else { do Y }` will always do Y
  - `while (true) { do X }` will keep doing X forever
  - `while (false) {do X}` will never do X
- They are most interesting when you use variables and boolean operators to calculate a boolean value which depends on the rest of the program (or the user's input etc.)

Boolean operators

- Boolean operators are operators which calculate a Boolean (true/false) result
- There are two kinds, comparison operators that work on numbers (mainly), and logical operators which combine other boolean expressions
- The main comparison operators are: == (equal to), != (not equal to),
  < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to)
- The main logical operators are: && (and), || (or), ! (not)

Examples of boolean operators

- `x < 2` – true if the variable x has a value which is less than 2
- `y == 6` – true if y is 6
- `2.00 ==  z` – true if z is (exactly) 2.00

- `x != 2` – true if x is not 2
- `! (x == 2)` – true if x is not 2
- `x < 2 && y == z` – true if x is less than 2 and y is equal to z
- `x >= 1 && x <= 10` – true if x is between 1 and 10
- `x < 1 || y < 1` – true if x is less than 1 or y is less than 1 (or both)

Notes on expressions

- We have been talking about numbers as if they are one kind of thing, but actually in a computer, they aren't – there are ints (integers), floats, doubles, chars, shorts and longs (and more). We will talk about these different types in a moment, and look at them in more detail later in the module.
- An expression represents a value. Many complex expressions combine two values to make a new value. That value may be of the same type (eg two numbers multiplied gives another number) or a different type (as in 6 == 4, which takes two numbers and returns a boolean value - false).
- Complex expressions in Java code are actually themselves instructions to the computer to do the calculation. So

  ```
  System.out.println("Hello "+"World");
  ```

  is actually two instructions: join "Hello " and "World" into one string, and then print that string.

Types

Java types

- We have mentioned from time to time that things in Java are of different types
- This is not just a casual usage (as in different kinds, or different sorts) – types are a specific and important part of Java
- Every piece of data that you use in a Java program has a type
- Every expression has a type, and every variable has a type
- And Java expects us to keep track of types and use them carefully and tells us off if we get it wrong (Java is called a ‘strongly-typed’ language)

Types we have seen already

- We have seen a few types in our examples already:
  - String – the type of string objects
  - String[] – the type of string arrays (lists of strings)
  - int – the type of whole numbers (integers, in maths)
  - boolean – the type of the values true and false, used in tests
  - Scanner – the type of a library to help us get user input

Types in Java

There are four kinds of type in Java:

- Primitive types – these are types for different sorts of numbers, and for booleans, which are the building blocks of larger data objects
- Classes – these are types associated with class definitions which define their state (what data they hold) and behaviour (what you can do with them), and are the basis for programming in Java. The main examples we have seen so far are the String and Scanner classes.
- Arrays – these are types which represent lists of objects of another type, as we have already seen in our examples with String[].
- Generic types – these are types which are built up out of other types (like arrays, but more general). We will talk more about them later in the course.

Primitive types

- Java has eight primitive types:
  - `byte    – 8 bit integer`
  - `short   – 16 bit integer`
  - `int     – 32 bit integer`
  - `long    – 64 bit integer`
  - `float   – 32 bit floating point (decimal)`
  - `double  – 64 bit floating point (decimal)`
  - `boolean – true/false`
  - `char    – 16 bit Unicode character`

- Primitive types mostly support numbers at different precisions
- They distinguish between integers (whole numbers) and floating point (decimals)
- For now, we will mostly be concerned with int and double numbers.
- boolean values and characters (char) are also primitive.

Class types

- A class type is created when a class is defined in a program.
- There is a large library of pre-defined classes (such as String and Scanner).
- Class names conventionally start with a capital letter, to distinguish them from variables (but notice that primitive types do not)
- We will learn more about how to make objects using classes soon – the only example we have seen so far is making a Scanner object with:

```
myInput = new Scanner(System.in);
```

Array types

- Arrays make lists of other things (for example a list of Strings or ints)
- An array has its own type depending on what kind of thing it is a list of
- So while String is a single string object, String[] is a list (array) of string objects
- And int[] is an array of ints
- You can always tell when you have an array type because it uses […] to identify it and to access data inside it.

Arrays we have seen already

- Our first array examples looked like this:

```
String[] places = {"London", "Brighton", "Paris"};
```

- And we used them in for loops like this:

```
for (String place: places) {
    System.out.println(place);
}
```

Generic types

- The last kind of type that you see in Java is the generic types
- We will talk about these in more detail later in the module
- But we will mention one example now, and you may see others
- Just as you can identify arrays because they have square brackets ([..]), you can identify a generic type because it has ‘pointy brackets’ (< … >)
- Between the pointy brackets you write one or more other types
- For example, ArrayList<String> is a generic type. It’s a bit like an Array only more useful, and we will introduce it in

Declaring and assigning to variables

Variables and types

- In Java, a variable can only contain values of one type
- Whenever you want to start using a new variable, you have to tell Java its name, and what type of value it can hold
- We call this declaring the variable
- And once a variable has a type, you can only store values of that type in the variable

Variable declarations

- In its simplest form a variable declaration looks like this:

```
type variable;
```

- For example we can say

```
String home;
int x;
Scanner myInput;
```

- Variables always have values, so in these examples the variable will set ('initialised') to a default value (0 for integer types, 0.0 for floating point, null for class types).

Variable declarations with initialisation

- We can also initialise our variable in the same statement:

```
type variable = expression;
```

- `expression` is an expression which calculates a value of the specified type.

```
String home = "Brighton";
Scanner myScanner = new Scanner(System.in);
```

- Initialising is generally good practice, to make sure that your variables have an appropriate initial value for your particular application.

Variable assignment statements

- After a variable has been declared (and maybe initialised), you may want to change its value at some point in your program. You do this with an assignment statement:

```
variable = expression;
```

- Again, the expression must calculates a value of the type specified in the variable declaration.

Notes on assignment statements

- An assignment statement looks like an initialisation statement, but without the type specifier. Don’t mix them up, however, because you may only declare a variable once.
- A common editing error is to copy a declaration statement and paste it in where you want an assignment. This results in an error because you are re-declaring your variable.

```
String room = "hall";
// later in the code
String room = myInput.next();
// Java will complain because it thinks room is being declared again
```

- What you need is just

```
room = myInput.next();
```

Notes on assignment statements

- Assignment statements also look like equations, but they are not. They are instructions to (a) evaluate `expression` and (b) assign its value to the `variable`. This is important because the expression may (often) include a reference to the variable itself, for example:

```
n = n+1;
```

- As an equation, this is nonsense. As an instruction it says, take the old value of `n`, add one to it, and save the result as the new value of `n`

Summary

- Variables let us save values and give them names, which allows us to use them in more than one place in a program
- Expressions let us write down/calculate values of different sorts (strings, numbers, booleans etc.)
- Variables, expressions and values have types
- In Java, whenever you make want to use a new variable, you have to tell Java what type it is, we call this declaring the variable
- And once a variable has a type, you can only store values of that type in the variable (approximately)
- This means that you can only use an expression of that type in an assignment statement for the variable

Lab exercises Week 1.04

Lab exercises – BlueJ

- Create a folder for this week's work on your S: drive eg at S:\CI401\week1.04
- Download BlueJ project week1.04-lab.zip from My Studies into this folder
- Open BlueJ on your computer and create a new project from the zip file in your new folder
- BlueJ will show you a folder full of Example files and Lab exercises

Lab exercises – coding

- Lab1/BuyingPaint – an exercise in declaring, inputting and using variables of different types
- Lab2/RunningTotal – an exercise in using a variable which keeps updating its own value, with a challenge task to look at RepeatForever loops
- Lab3/FortuneTeller – an exercise in constructing different boolean tests in a fun application to tell your fortune
- Lab4 – additional lab to write your own quiz like FortuneTeller
- SimpleExpressions and BIDMAS (from the lecture) are also there if you want to look at them more closely

Lab exercises – we are here to help!

- **If you get stuck, ask for help!**
- **Even if you don’t get stuck, talk to us!**