## Primitive data types

| | |
|---|---|
| int | Main type for storing whole numbers/integers. |
| float | Denotes real numbers with a smaller range and precision. Sufficient for storing 6 to 7 decimal digits. Precision is about 15 decimal digits. |
| double | Main type for real numbers. Sufficient for storing 15 decimal digits. |
| boolean | Stores only two possible values: true or false. Used to represent any binary situation, used mainly for recording decisions. |
| char | Stores a single character/letter. Represents all types of characters, including: Letters, formatting characters, special characters, and characters in other languages. Characters are written in single quotes. |
| byte | Stores whole numbers from -128 to 127. |
| short | Stores whole numbers from -32,768 to 32,767. Can store only less than int. |
| long | Stores whole numbers from about -(9e+18) to (9e+18). Can store more than int. |

**Primitive data types:** A primitive data type specifies the size and type of variable values, and it has no additional methods.

## Comments

Single line comments:

```
// Comment1
code // Comment2
```

Multi-line comments:

```
/*
* Comment
* Continuing comment
*/
```

**Comments:** Comments are ignored by the computer, they are removed during compilation and exist simply to make the code easier for people to understand.

## General class definition and body

```
public class ClassName
{
          Fields
          Con str uctors
          Methods
}
```

**Classes:** Classes describe all objects of a particular kind, and determine the fields, constructors, and methods these specific instances will all have.

**Class names:** By convention, class names start with an uppercase letter (to distinguish from other names like variables and methods.**

**Classes and types:** A class name can be used as the type for a variable. Variables that have a class as their type can store references to objects belonging to that class

## General constructor definition and body

```
public class ClassName
{
      Fields omitted
           public ClassN ame -
(pa ramtype paramname, etc.)
         {
                        fie ldName =
fieldV alue1;
                        fie ldName2
= fieldV alue2;
                        fie ldName3
= paramname;
               etc. ---
         }
      Methods omitted
}
```

**Constructors:** Constructors are responsible for ensuring that an object is set up properly when it is first created/that an object is ready to be used immediately following its creation.

**Initialisation:** This construction process is also called initialisation. The constructor initialises the fields.

**Note:** In Java, all fields are automatically initialised to a default value if they are not explicitly initialised (0 for integers etc.)

## General while loops

```
while (boolean condition) {
      loop body*
}
```

**While loops:** A form of indefinite iteration loop.

**Note:** While the condition evaluates to true, then the body is executed; and once it evaluates to false, the iteration is finished

**Note:** The condition could evaluate to false on the very first time it is tested. If that happens, the body won't be executed at all.

**Note:** The while loop does not need to be related to a collection. Even if processing a collection, we do not need to process every element.

## General do-while loops

```
do {
      loop body
} while (boolean condition);
```

**Do-while loops:** A form of indefinite iteration loop, but loop is executed at least once.
**Note:** While the condition evaluates to true, then the body is executed; and once it evaluates to false, the iteration is finished.
**Note:** The while loop does not need to be related to a collection. Even if processing a collection, we do not need to process every element.

## Other keywords

| | |
|---|---|
| void | Methods have return types that specify what type of data they return. If a method does not return any specific data, the return type is 'void'. |
| null | Object variables don't always refer to actual objects. When an object variable is first declared, it is initialised to the special value null, which means that the variable isn't pointing to an object. |
| this | In a 'name overloading' situation where the same variable name is being used for 2 different entities in the same scope, the variable referenced will be the closest defined. To reference the field instead of the paramter write 'this.' before the name. |

[Note to self: these may be sorted later into other groups as I cover more similar material]

## Casting

```
(newdatatype) value
```

**Casting:** Casting means to change a value from one type to a "corresponding" value in another type.
**Note:** We can cast char values to their Unicode int values and vice versa.

## Math

| | |
|---|---|
| Math.abs | Absolute value |
| Math.pow | Raise to the power/ exponents |
| Math.max; Math.min | Find the maximum; Find the minimum |
| Math.sin; Math.cos; Math.tan | Trigonometry |
| Math.round | Round number |
| Math.PI; Math.E | Use constants of Pi and E |

**Math:** Math defines pretty much all mathematical functionality that you will ever need.
**Note:** It has other methods that are not stated here.
**Note:** All of the methods in Math are static methods.

## Changes to parameters, equality over objects

| | |
|---|---|
| Method with primitive type parameter | Updates to that parameter are local only. |
| Method with object type parameter (e.g. array) | Parameter is a new variable but refers to the same object; changes are persistent. |

## Changes to parameters, equality over objects (cont)

| | |
|---|---|
| Equality over arrays and objects | True if the variables point to the same object, false if not even if the contents of the objects are the same. |
| java.util.Arrays.equals(object1, object2) | True if the variables point to the same objects and also if the contents of different objects are the same. |

## Final variables

```
private final datatype variableName = variableValue; OR
private static final datatype variab leName = variab leValue
```

**Final variables:** Final variables must be initialised immediately and can never be changed.

## Throwing unchecked exceptions

```
public returntype methodName(paramtype paramvalue etc.) {
      code body
          if boolea n_e xce ption {
                    throw new Except ionName ("ex ception messag e")
          }
      code body
}
```

**Unchecked exceptions:** When an exception is thrown and it is an unchecked exception, the system halts with an error message. It is a standardized way to deal with errors to provide informative feedback.
**Error type:** Used with client code is seriously wrong - attempts to use your methods incorrectly by passing incorrect parameter values.

## Object data types

| | |
|---|---|
| String | Used to represent a sequence of characters, including: names, addresses, general text etc. Strings are written in double quotes. |
| ArrayList | A Java class from the java.util package. |
| Random | A java class from the java.util package |

**Object data types:** Data types that are actually objects, which contain methods that can be called to perform certain operations on them.

## Logical operators

| | |
|---|---|
| a && b | a and b are both true (and). |
| a \|\| b | At least one of a and b is true (or). |
| !a | a is false/not a (not). |

**Precedence:** && has a higher precedence than ||.

## Arithmetic operators

| | |
|---|---|
| + | Addition - adds together two values. |
| - | Subtraction - subtracts one value from another. |
| * | Multiplication - multiplies two values together. |
| / | Division - divides one value by another. When dividing integers the remainder is truncated (integer division); when dividing doubles the return value is exact. |
| % | Modulus - returns the remainder of dividing the one value from another. |

## Arithmetic operators (cont)

| | |
|---|---|
| ++ | Increment - increases the value of the following variable by 1. |
| -- | Decrement - decreases the value of the following variable by 1. |

**Arithmetic operators:** Used to perform common mathematical operations.
**Precedence:** Precedence and associativity are as normal as in maths.

## General method definition and body

```
public class ClassName
{
      Fields and constr uctor
omitted
          public returntype
method Nam e(p ara mtype1
paramn ame1, etc.)
         {
                      sta tem -
ent1;
                      sta tem -
ent2;
           Body continued
         }

     Other possible methods
omitted
}
```

**Methods:** Objects have methods that we use to communicate with them. We can use a method to make a change or to get information from the object.

## Conditional statements/ if-else-statements

```
if (perform some test) {
     do these statements if the
test gave a true result
}
else if (perform some test) {
     do these statements if the
if-sta tement and else if
     -state ments above
returned false,
> but the test for this statement returned
true.
}
else {
    do these statements none of the above
tests returned true
}
```

**Note:** It is possible to have only 1 if-statement and no else if or else statements. There can also be many if-statements and else if statements in the same block.

## Importing library classes

```
import librarypackage.ClassName;
OR
import librar ypa ckage.*
```

**Note:** Usually happens at the very top of the program.
**Note:** Using * means that all classes in that package is imported.
**Note:** Some library classes are imported automatically, including Math, String, Integer, Character, Boolean etc.

## Creating objects

Object creation:

```
objectName = new ClassN ame (pa ra
c.);
```

Creating object for field:

```
private ClassName fieldName;
...
fieldName = new ClassN ame (pa ram
.);
```

**Note:** Done in the constructor. Can also create an object assignment to a field, making the field point to the object, but the field variable will have to be declared first
**Note:** If you haven't called 'new', you haven't created an object.

## General for-each loops

```
for (elementType element :
collectionName) {
     loop body
}
```

**For-each loops:** A type of definite iteration.
**Note:** For each element in collection, execute loop body.
**Note:** The new local variable ('element') used to hold the list elements in order is called the 'loop variable' (any name possible). The type of the loop variable must be the same as the declared element type of the collection.
**Note:** We cannot change what is stored in the collection while iterating, but can change the states of objects already within the collection.

## General for loops

```
for (initialisation; boolean
condition; post-body action) {
     loop body
}
```

**For-each loops:** A type of indefinite iteration.
**Note:** For each element in collection, do the things in the loop body.
**Conditionals:** Conditionals/if-statements can be used in loops.

## General JUnit test

```
@Test
public void testme thod() {
     setup code
          ass ert Equ als (pa -
ram values)
}
^
assert Equ als (er ror Mes sage,
correc tRe turn, method call) OR
```

## General JUnit test (cont)

> assertEquals(errorMessage, correctReturn, methodcall, double) OR assertTrue(errorMessage, correctReturnBoolean) OR assertFalse(errorMessage, correctReturnBoolean)

**JUnit tests**: JUnit classes run your code and compare actual results with expected results.
**Note**: Some limitations are that - printing can't be tested; can test only changes to an objects state/values returned by methods; can test only 'public' methods; can't see inside methods.

## Static methods

```
public static returntype
methodName(paramtype parameter
etc.)
```

**Static methods:** In static methods, the values returned don't depend on the state of an object, only on the arguments provided i.e. you can call a static method without creating an object first. (In fact it is sometimes impossible to create an object)
**Note:** They are sometimes called class methods.
**Note:** Can be invoked with the class name, rather than an object name.

## Dealing with arrays

Declaring arrays:
`datatype[] arrayName;` OR
`datatype[] arrayName = {varia ble1, variable2, etc.};` (array literals)
`dataty pe[][] arrayName`: etc. for 2d, 3d + arrays
Creating arrays:
`arrayName = new dataty pe[ noO fVa ria bles];`
`arrayName = new dataty pe[ ]{v ari able1, variable2, etc.}` (array literals)
Size of array:

## Dealing with arrays (cont)

```
arrayN ame.length
```

Referencing elements:

```
arrayN ame [index]
```

**Arrays:** Arrays are fixed-size collections that can store object references or primitive values. It is an indexed sequence of variables of the same type.
**Note:** The variables do not have individual names.
**Referencing elements:** Elements can be used in the same ways and in the same contexts as any other variable of that type.
**Note:** Arrays can share memory - 'Aliasing'.
**Objects:** When using an arrays with elements of object type, you also have to populate the array with a loop.

## Making assertions

```
code body
assert boolea nCo ndition : " -
str ing ";
code body
```

**Assertions:** A debugging mechanism to use when you are developing complicated code. When the assertion is executed, the boolean condition is evaluated. If it is true, execution continues. If it is false, execution is halted with an (unchecked) AssertionError, and the message string is printed.
**Error types:** Logic errors. Check what values a given variable has compared to what it should have.

## Relational/Comparison operators

| | |
|---|---|
| == | Equal to |
| != | Not equal to |
| < | Less than |
| <= | Less than or equal to |
| > | Greater than |

## Relational/Comparison operators (cont)

| | |
|---|---|
| >= | Greater than or equal to |

**Relational operators:** Operators used to compare two values - usually numbers, but also sometimes other types.
**Precedence:** All have lower precedence than all arithmetic operators, and higher than all logical operators.

## Augmented assignment/ Assignment operators

| | |
|---|---|
| a += b; | Equivalent to a = a + b; |
| a -= b; | Equivalent to a = a - b; |
| a *= b; | Equivalent to a = a * b; |
| a /= b; | Equivalent to a = a / b; |
| a %= b; | Equivalent to a = a % b; |
| a &= b; | Equivalent to a = a && b; |
| a != b; | Equivalent to a = a \|\| b; |

**Augmented assignment:** Java supports augmented assignment for common arithmetic and logical operators.

## General field definition and body

```
public class ClassName
{
          private type
fieldName;
          private type2 fieldN -
ame2;
         etc.

     Constr uctors and methods
omitted
}
```

**Fields:** Fields store data persistently within an object, that have values that can vary over time. Also known as instance variables. Every object will have space for each field declared in its class.

## General variable declaration and assignment

Field declaration (see other cheat block):

```
private type variab leName;
```

Local variable declaration:

```
type variab leName;
```

Assignment statement:

```
variab leName = newValue;
variab leName = 2 * variab leName;
variab leName = variab leName ** 2 +
5;
```

Shorthand (declaration + assignment):

```
type variab leName = newValue;
```

**Variables:** The basic mechanism by which data is organised and stored (long-term, short-term, and communication etc.). Variables must be declared before it is used.
**Variable names:** Variable names should always start with a lower-case letter.
**Local variables:** A local variable is defined inside a method body, as opposed to a field variable that is defined outside the method and a parameter that is always defined in the method header.

## Dealing with Strings

| | |
|---|---|
| "Stringa" + "Str-ingb" -> "Str-ingaStringb" | String concatenation, achieved with the + operator |
| System.out.p-rint("String") OR System.ou-t.println("Strin-g") | String printing. println enters to a new line at the end in addition to displaying the string. |
| \n | Prints to a new line. Also known as 'carriage-re-turns' |
| \t | Prints the tab character. |

## Dealing with Strings (cont)

| | |
|---|---|
| String.to LowerC-ase(); String.to UpperC-ase(); | Changes all characters in the string to lowercase/uppercase. |
| String.le ngth() | Returns the number of characters in the string. |
| String.ch arAt(i-ndexnu-mber) | Returns the character at the given index. |
| String-1.comp-areTo(-String2) | Compares 2 strings. It returns a negative number if the target comes before the argument, a positive number if the target comes after the argument, 0 if they are equal. |

**Strings:** Strings are used to represent a sequence of characters, including: names, addresses, general text etc. They are written in double quotes.
**Note:** String is a class defined in the library
**Note:** Strings in Java are immutable objects (they cannot be changed after they are created).
**Ordering:** Ordering is by the first letter in which they differ, otherwise by their length. (Note that it is based on Unicode values - not save for case and punctuation etc.)

## Method calls

Internal method calls:

```
method Nam e(p ara mva lue1, paramv alue2, etc.)
```

External method calls:

```
object Nam e.m eth odN ame (pa ram value1, paramv alu
e2, etc.)
```

**Internal method calls:** When an object calls a method on itself. In this case the object name will not need to be specified.

**External method calls:** When one of the methods of the object in turn calls a method of another object to do part of the task. In this case the object the method is called on needs to be specified.

## Dealing with ArrayLists

Declaring ArrayList field variables:

```
private ArrayL ist <ty pe> variab leName;
```

Declaring ArrayList local variables:

```
ArrayL ist <ty pe> variab leName;
```

Creating the collection:

```
variab leName = new ArrayL ist <>();
```

or `variab leName = new ArrayL ist <ty pe>();`

'add' operation:

```
variab leN ame.add(expression of the approp riate typ
e);
```

'size' operation:

```
variab leN ame.si ze();
```

'get' operation:

```
variab leN ame.get(index);
```

'remove' operation:

```
variab leN ame.re move(index);
```

**ArrayList:** A Java class from the java.util package. Dynamically sized collection, can store both object references or primitive values.

**Operations:**

- add: Adds the written object to the end of the collection.
- size: returns an int of the collection size
- get: retrieve an item from a specified index
- remove: removes an item from a specified index. Will move up the indices of items behind it.

[- others: search online]

## Chaining method calls

```
objectName.methodName1().methodName2().methodName3();
```

**Note:** This looks as if methods are calling methods, but the chain of method calls must be read strictly from left to right.

## Access modifiers

Public modifiers:

```
public datatype variab leName; OR
public returntype method Nam e(p ara mete
rs)
```

Private methods:

```
private datatype variab leName; OR
private returntype method Nam e(p ara met
ers)
```

**Public methods/variables:** Public fields, methods and constructors in a class can be accessed/invoked from any class in the program.

**Private methods/variables:** Private fields, methods and constructors in a class can be accessed/invoked only from the class where it is defined.

## Enumerated types

Creating enum class:

```
public enum ClassName
{VALUE 1,V ALUE2, VALUE3, etc...}
```

Accessing values:

```
ClassN ame.VALUE
```

Built-in facilities:

Compared for equality and inequality: `variable1 == variable2`

Added to strings: `" Str ing " + variable`

Processed using for-each loop: `for (ClassName item : Cl
assN ame.va lues()) {...}`

Ordered using ordinal(): variable.ordinal()

**Enum:** Used to represent discrete data with only a small number of possible values.

## Throwing checked exceptions

```
public returntype
methodName(paramtype paramvalue
etc.) throws ExceptionName {
        code body
        if boolea n_e xce ption
{
                throw new
Except ionName ("ex ception
messag e")
        }
        code body
}
// Client code:
accessmod returntype anothe -
rMe tho d(p ara mtype paramvalue
etc.) {
        try {
          code body (for when
everything is fine)
        }
         catch (Excep tio nName
e) {
          code for when things
go wrong
        }
         catch (Excep tio nName
e) {
          code for when things
go wrong
}
```

**Checked exceptions:** When an exception is thrown and it is a checked exception, the system tries to find some object in the program able to deal with it without crashing

**Note:** More complicated than unchecked exceptions - method is required to declare it might throw exceptions; and client code is required to provide code that will be run if so

**Error type:** Situations that are not entirely unexpected and from which clients may be able to recover.