3.1) Compilers convert source code into machine code which is then executed by the processor whereas interpreters translate and execute the source code line by line. Compliers produce the compiled file which is stored and can be rerun without the need to recompile the source code, however every time Interpreted code is run, the source code must be translated. Compilers are generally faster than Interpreters as interpreters require a lot of overhead when translating code line by line. To contrast, complied code is unique to each platform as machine code differed between platforms.

3.2) Java is known to be a secure language because it implements many features to avoid security problems. Firstly, code is compiled into bytecode which goes on to e run by a virtual machine, the use of the virtual machine means the program cannot accidentally access resources or secure memory blocks of the main computer preventing crashes or potential data theft. Secondly, Within the virtual machine, java uses automatic memory management and garbage collection (deletes unused variables and object to save memory) to prevent filling up the memory and slowing down a computer.

Java is platform independent because it is a compiled language which produces bytecode. As bytecode is made up of binary, this code will be unique to specific platforms as different platforms would have a different instruction set. Therefore one machine wouldn't be able to understand java code compiled for another machine.

3.3) Variables are data items which can be referenced by a name, their value can be changed during the execution of a program. Similarly, Constants are also data items which can be referenced by a name, however constants values cannot be changed as a program runs. Once a constant is declared and given a value, this value can only be accessed, but not changed.

3.4) 8 Primitive data types in java are:

1) Integer (int) java code→
```java
int myInt = 5;
```

2) Long (long) java code →
```java
long myLong = 100000000000000;
```

3) Short(short) java code →
```java
short myShort = 123;
```

4) Byte(byte) java code →
```java
byte myByte = 99;
```

Integer, Long, Byte and Short types are all used to store whole number, however the difference lies between the storage space they use and therefore the range of numbers that these types can store. Bytes use 8 Bits, Shorts use 16 Bits, Integers use 32 Bits, and Longs use 64 Bits.

5) Double(double) java code →
```java
double myDouble = 4.321;
```

6) Float(float) java code →
```java
float myFloat = 1.234;
```
Double and Float types are both used to store decimal numbers however Doubles use 64 Bits and Floats use 32 Bits. Doubles can therefore store a greater range of numbers that a float, or can have a higher degree of precision that a float.

7) Char(char) java code →
```java
char myChar = 'N';
```

Char Types are used to store individual characters, they are shown inside single quotation marks. Chars use 16 bits in java.

8) Boolean(boolean) java code →
```java
boolean myBool = true;
```

Booleans are used in java to represent 'true' or 'false', they only require one bit.

3.5) Casting is the process of changing the data type of a value in a variable. Implicit casting is then the transfer from one data type to another can be done without losing information. If casting has a risk of loosing information, then the process is known as explicit casting.

3.6) Overflow is the occurrence when there aren't enough bits to store the data required. It can happen after a mathematical calculation where the result is too large to be stored as an integer, e.g casting an integer of the value 5000 into a byte data type. The byte type only has 8 bits which isn't enough to store the value 500

3.7) There are 4 main features of OOP:

1) Inheritance → Inheritance is the concept of deriving a class from other classes. A child class can inherit attributes and methods from its parent class and this is a way to reuse code and make a program more maintainable. Inheritance is a way to build upon existing classes.
2) Polymorphism → Polymorphism is where the same action can be preformed in multiple ways. It can be carried out in 2 ways. Method Overloading, which is compile time, where 2 methods of a class exist under the same name, however they take in different arguments and therefore can be identified. The second way is Method Overriding, which is done during runtime. This method is associated to inheritance where a method defined by the parent class has the same name as a method defined by a child class. In this case the method defined by the child class is used.
3) Encapsulation → Encapsulation is the idea of wrapping attributes and methods into a single unit (Object). This allows us to achieve 'data hiding' and keep data unknown to the outside code. By declaring attributes or methods 'private' these fields cannot be seen by anything outside

of an object and therefore getters and setters are required to view or change them. PTO FOR 4

4) Abstraction → Abstraction is where we hide all irrelevant data in order to reduce a programs complexity and increase its efficiency. This can be done by using abstract classes or interfaces.