Classes and Objects

Introduction to OO Programming

Introduction

- Computer programming involves writing instructions with the goal of solving a specific problem.
- Traditional programming languages involve the programmer defining and solving the problem in terms defined by the computer. Object oriented programming languages allow the programmer to define problems in terms of the problem
- The real power of a computer language comes from its ability to define complex, multipart, structured data types that model the complex properties of real world objects

Object types

- Every value used in a Java program has a type.
- So far we have used primitive data types in our programs.
- Object oriented programming languages allow the use of both *primitive* and ***Object*** types.
- A Java programmer can create their own Object types, or can create Objects from types defined by others.

State and Behaviour

- An `Object` can store data, and can also perform operations.
- An `Object` is said to have **State** and **Behaviour**.
- **State** is the things an `Object` knows. **Behaviour** is the things an `Object` does.
- This is built around the concept of objects in the real world. For example, a car has state (colour, size, no. of doors, fuel level) and behaviour (drive, break, stop, add more fuel).

Instance Variables

- State is represented in the form of **Instance Variables**.
- This is the information that an `Object` knows.
- For example, a `BankAccount` Object might have instance variables `accountNumber` and `balance`.
- A `Student` Object might have instance variables `studentID` and `studentName`

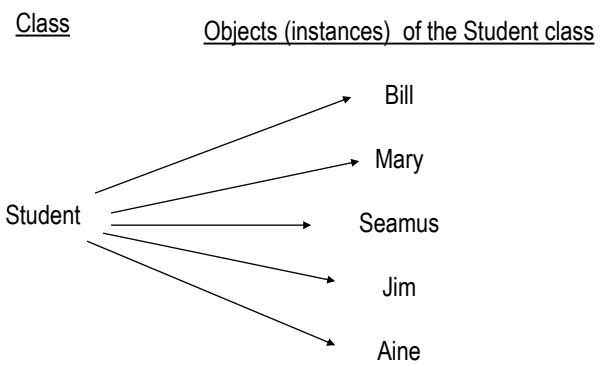
Methods

- Behaviour is represented in the form of **Methods**.
- A method is an operation which the object can perform, usually by accessing or manipulating its instance variables.
- For example, a `BankAccount` Object might have `getBalance()` and `withdraw()` methods.
- These methods perform operations on the instance variables of that Object.

Objects and Classes

- An Object is an entity that you can manipulate in your program.
- An Object is created from a template known as a Class. Each Object is an **instance** of a Class.
- For example, **`keyboardIn`** is an instance of the `Scanner` Class.
- Once a Class is implemented (or written), you can create as many Objects of that Class as you like.

Objects and Classes

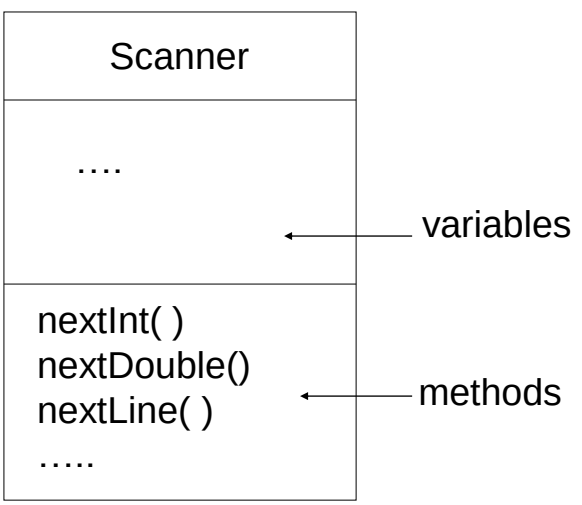


Declaring an Object

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

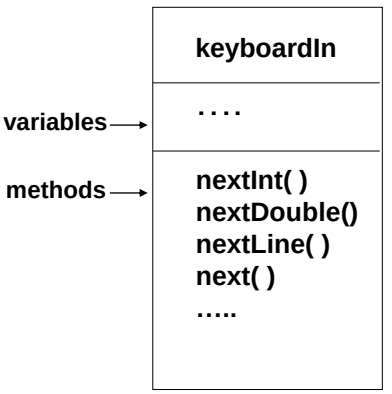
- This line of code declares an **object** called **`keyboardIn`**, which is an instance of the **`Scanner`** **class**
- The object **`keyboardIn`** has all the variables and methods defined in the **`Scanner`** class.

The Scanner class



A Scanner object

```
Scanner keyboardIn = new Scanner(System.in);
int age;
age = keyboardIn.nextInt();
```

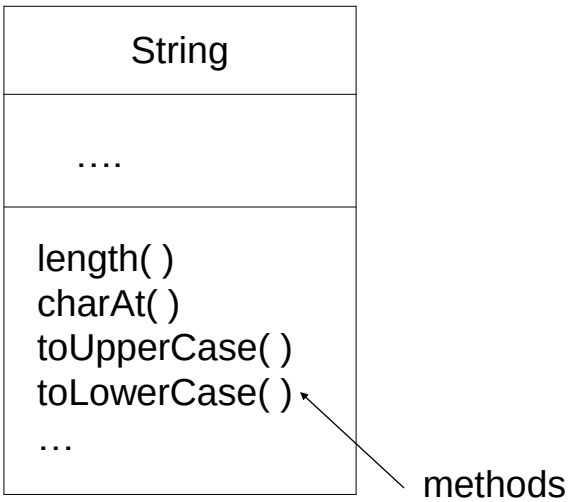


The String class

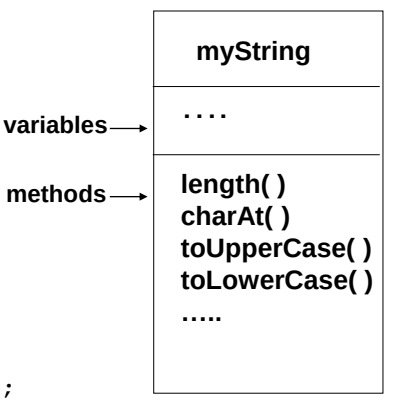
```
String name = new String();
name = "Michael Murphy";
```

- A String is a sequence of characters which is enclosed in quotation marks.
- Here, `name` is an object which is an instance of the `String` class.
- `name` has all the variables and methods defined in the `String` class.

The String Class



A String object




```
int numLetters = 0;
String myString = new String();
myString = "Hello World";
numLetters = myString.length();
System.out.print("No. of chars:"+numLetters);
```

Calling Methods

- The programmer calls the methods of a class on a particular object to perform operations on its instance variables.
- For example, call the `nextInt()` method of the `Scanner` class on `keyboardIn` (object of type `Scanner`).

```
Scanner keyboardIn = new Scanner(System.in);
int studentAge = 0;
studentAge = keyboardIn.nextInt();
```

- It is important to understand how to provide inputs into a method, and how to obtain the output from a method.

Method parameters

- Some methods can perform their operations without any inputs (parameters).
  - the `length()` method of the `String` class does not need any input. It can simply be called as is.

    ```
    int noOfChars = name.length();
    ```

  - the `nextInt()` method of the Scanner class.

    ```
    num1 = keyboardIn.nextInt();
    ```

  These method take no arguments (parameters)

Method parameters

- Other methods require information from the programmer in order to work. (take arguments (parameters)
- For example, in order to use the `charAt()` method of the `String` class, you need to provide the position from which you require the character.

```
String name = "Joe Bloggs";
char letter = name.charAt(2);
```

Method parameters

- An input for a method is known as a **parameter** (or argument)
- Methods are written to expect parameters of a certain type. For example, the `charAt()` method of the `String` class expects an `int` value as a parameter.
- The programmer must use the right type when passing parameters to a method. Failure to do this will result in either a syntax error, or unexpected results.

```
public class StringTester
{
  public static void main(String [] args)
  {

      String myName = new String();
      myName = "Joe Bloggs";

      char firstLetter;

      firstLetter = myName.charAt(0);

      System.out.print("First initial " + firstLetter);

  }
}
```

Method parameter

```
public class StringTester
{
  public static void main(String [] args)
  {

      String myName = new String();
      myName = "Joe Bloggs";

      char firstLetter;

      firstLetter = myName.charAt("J");

      System.out.print("First initial " + firstLetter);

  }
}
```

Invalid Method parameter
This will cause a Syntax error

Method return values

- A **return value** is a value which is returned as the result of a call to a method.
- Some methods perform operations and do not return any information. For example, the `println()` method of the `System.out` object simply performs an operation. It does not return any information.
- Other methods return information when invoked.
- For example, the `length( )` method of the `String` class returns a value representing the length (no of chars) of the `String`.

Method return values

- Methods will return values of a certain type. For example, the `charAt( )` method of the `String` class returns a **char** value.
- The programmer must write code to deal with the value that a method returns (otherwise it is lost).

```
String myName = new String();
myName = "John Lennon";
int numLetters;
numLetters = myName.length();
System.out.print(numLetters  + " characters");
```

assign return value to variable of type int

Some String methods

Method	Input	Return type
`length()`	`None`	`int`
`charAt()`	`int`	`char`
`concat()`	`String`	`String`
`equals()`	`Object`	`boolean`
`equalsIgnoreCase()`	`String`	`boolean`
`toUpperCase()`	`None`	`String`
`toLowerCase()`	`None`	`String`
`compareTo()`	`String`	`int`
`startsWith()`	`String`	`boolean`
`endsWith()`	`String`	`boolean`