Java & OOP

4. Writing classes

Source File

- Has "`.java`" extension
- Layout

  Package statement

  Import statement(s)

  Class definition(s)

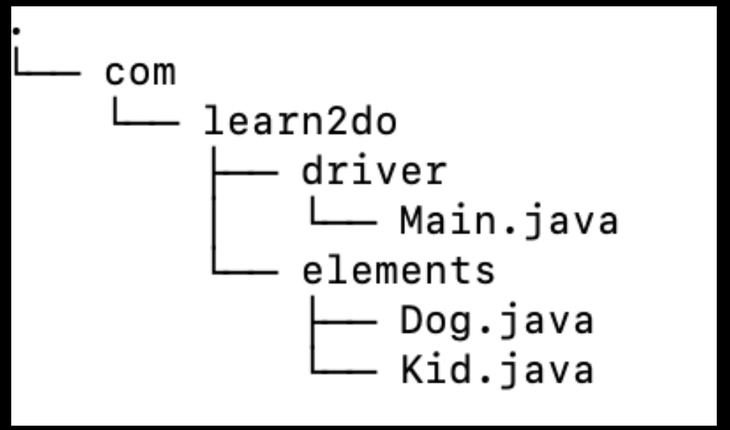
Package Statement

- Directory structure of the source files
- packages

```
com
com.learn2do
com.learn2do.elements
com.learn2do.driver
```

- In `Kid.java`

```
package com.learn2do.elements;
```



Import Statement

- While using another class in your source file
- To use `Kid` class
- Import with the path ()

```
import com.learn2do.elements.Kid;
```

Class definitions

- Syntax

```
[public] class <class name> {


}
```

- `public` is optional
- `public` class can be seen (used) by classes outside the package
- Non public class is internal to the package

class definitions and source file

- There can be more than one class definition in a source file
- But only one of them can be `public`
- Name of the source file is the name of the class
- Classes are mostly named after nouns
- `Kid` class resides in `Kid.java` and is `public`
- `Kid.java` is placed in `elements` directory

Kid class

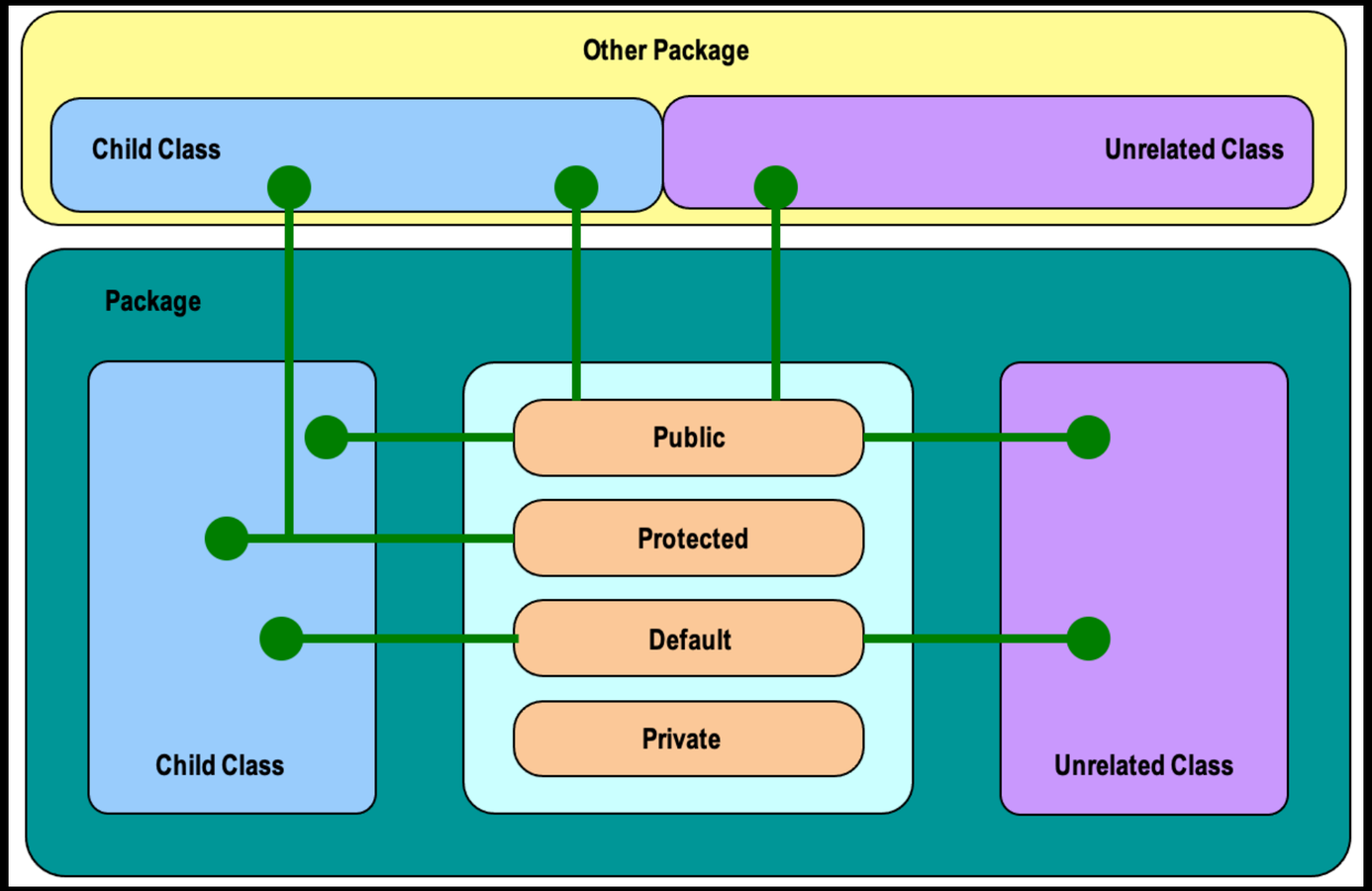
```
package com.learn2do.elements;

public class Kid {
  //definition
}
```

- There are no imports

Defining a class

- A class contains `state` and `behaviour`
- State/Property is defined using `variables`
- Behaviour is defined using `functions`
- You can control access to the `state` and `behaviour`
  - Controls access by other classes
  - Done using access specifiers `private, protected, public & default`

Access specifiers



State/Property

- Variable definition

```
<access-speicifier> [static] <type> <variable_name> [= value];
```

- Each object will have a copy of variables (non static - Instance), accessed using object variable
- `static` variables are shared by all objects of the class and accessed using the class

Varible initialization

- Varibles are initialised with default values
- You can intitialize with other than defualt values
- OK

```
public String name = "John";
public int age = 12;
```

- Not OK - avoid

```
public String name = null;
public int age = 0;
```

Kid class - Instance variables

```
public class Kid {
    public String name;
    public int age;
}
```

- `public` is used because `Main` class (in `driver` package) needs access

Defining Behaviour / Functions / Methods

```
<access-speicifier> [static] <type> <function_name>([parameters]) {
  //statements (body)
}
```

- `static` methods can access only `static` variables
- `non-static / instance` methods can access both `static & non-static` variables
- `type` of the function is the `type` of the data returned by the function
- If a function does not return anything `void` type is used

static function

```
public class Shapescalculator {
    public static double calculateTriangleArea(double base, double height) {
        double area = 0.5 * base * height;
        return area;
    }
}
//Somewhere else in another class
double triangle_area = Shapescalculator.calculateTriangleArea(10.0, 5.0);
```

- Works only with the parameters passed
- Does not access any instance variables
- Better be `static`

Functions

- Parameters while defining the function - formal parameters
- Parameter while calling the function - actual parameters
- Values from actual parameters are copid into formal Parameters
- Formal parameters are destroyed after function execution
- While passing object only the reference value is copied
- Hence strictly ***Call by Value*** only

Construction

- When a client creates an object using `new class_name()`
- A piece of code called `constructor` is invoked in the class
- definition

```
<access-specifier> <class_name> ([parameters]) {

}
```

- Why access specifier?
- No return type, Not a method & Not a member

Constructor Types

- default - no parameters, may not have any code

```
public Kid() { }
```

- Parametrized - with parameters and code to initialize instance variables

```
public Kid(String name, int age) {
    this.name = name;
    this.age = age;
}
```

- If you dont provide any constructor Java compiler will insert a default constructor *but will be withdrawn if you provide a parametrized constructor*

`static` block

- Executed while class is loaded for the first time
- Initialization code for all objects that are going to be created

```
static {
//statements
}
```

Accessors & Mutators

- You dont want the instance varible(s) to be public
- Accessor - The function used by a client to access the property
- Mutator - The function used by a client to set/change a property
- There can be methods that can mutate/change a property
- If a class has a private scope for all the instance variables and no Mutator methods provided then - **Immutable** class

Accessors & Mutators

- Accessor for `name` property (Note the cases)

```
public String getName() {
    return this.name;
}
```

- Mutator

```
public void setName(String value) {
    this.name = value;
}
```

Naming conventions

- package names - all lower case
- Class names - all word(s) capitalised
- Member names - First word all lower case, remaining capitalised

Next

Checking parameters