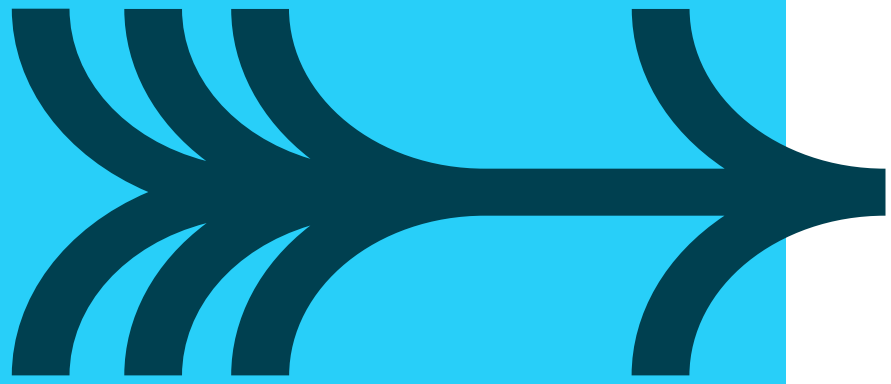
POWERING POTENTIAL

Types II – Fields and Methods

CONTENTS

- **Objectives**
  - To understand how to define the functionality of types
- **Contents**
  - static – what does it mean?
  - When to use a static field, property or method

- **Hands on Labs**



Static – what does it mean

Static means 'belongs to the class, not to an instance of the class

- Static members visible via the class name
- No requirement to instantiate the class, often a utility class

```
System.out.println("OK");
```

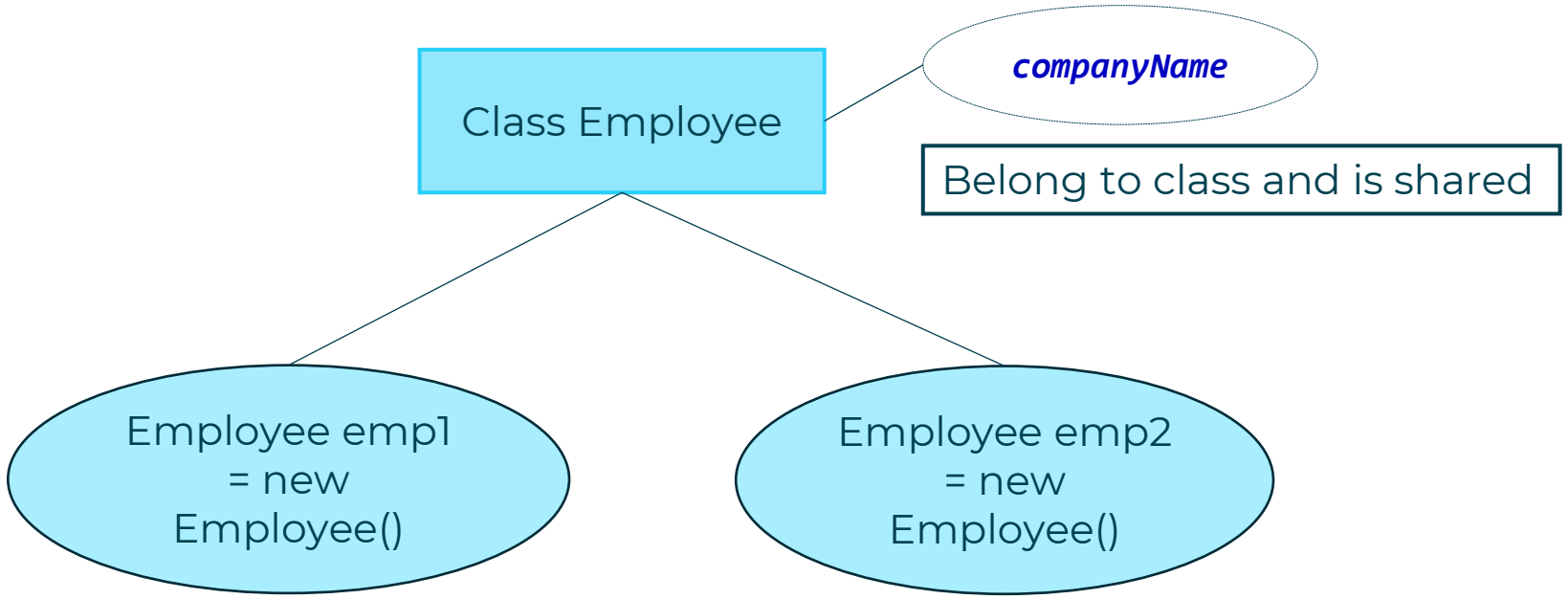
```
System.out.println(Math.PI);
```

```
System s = new System();
s.out.println();
```

```
Math m = new Math();
m.sqrt(25.6);
```

Defining Static fields

```
public class Employee {
    public static String companyName = "QA";

    private String name;
    private int age;
}
```



Using Static fields

```
public class Employee {
    public static String companyName = "QA";
    private String name;
    private int age;

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

```
public static void main(String[] args) {
    Employee emp = new Employee("Bob",25);

    Employee.companyName = "QA Ltd";
    emp.companyName = "QA Ltd";
}
```

✔

✘ Does not belong to an instance

Instance members – Methods

Instance methods operate on instances of a type

- Can't invoke them until you have an instance
- Can access instance fields and methods
- Can also access static fields and methods

```
public class Car {

  private int speed;
  private double totalCO2;

  public void accelerate( int weightOfFoot ) {
    speed += weightOfFoot;
    totalCO2++;
  }
   ...
}
```

Field has 'class' scope

```
Car car1 = new Car();
car1.accelerate( 5 );
Car.accelerate( 5 );
```

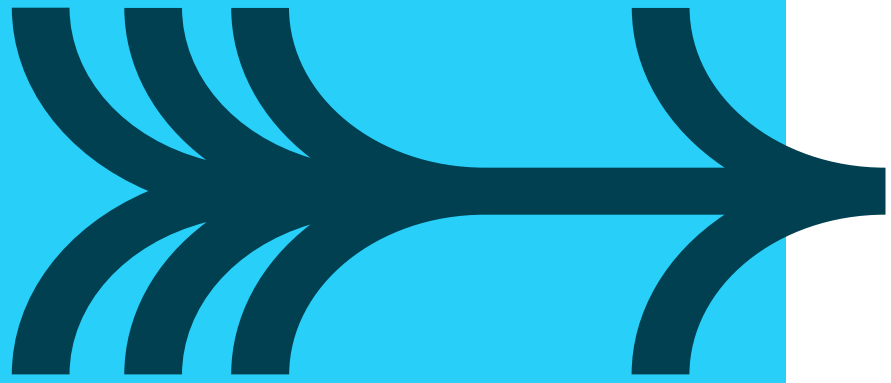
this

- **`this` refers to the object on which method was invoked**
  - Could write accelerate method as follows:

```
public class Car {

   private int speed;

   public void accelerate( int weightOfFoot ) {
      this.speed += weightOfFoot;
   }
   ...
}
```

  - In an ‘instance’ context there is always a ‘this’ *****
    - It is a reference to the object on which method was invoked
    - Think of it as a ‘hidden’ 1st parameter (of each instance method)

ENCAPSULATION



- **Types encapsulate state and "secret" behaviour**
  - Introduces *loose coupling* between code
  - Promotes parallel development
  - Improves testing
  - Facilitates re-use of components

- **You can controls accessibility using access modifiers**
  - Apply to type definitions - the class itself
  - But also to its members - fields and methods

Accessibility Modifiers

- **The following modifiers are available in Java**

Modifier	Description
`public`	Any code can (potentially) access the field/method
`private`	Only code within the type definition can access the element.

Accessibility Modifiers

- **The following modifiers are available in Java**

Modifier	Description
`protected`	Only code within the type itself, code in any type in same package and code in derived types can access.
`<no modifier>`	Only code within the same package can access the element

Accessibility of Types

- **Types can have their own access controlled**
  - **Top level types can only be `public or default`**

```
public class Car {
  private Engine e;
  ...
}
```

```
class Engine {
   ...
   ...
}
```

Note: Java compilers allow 'Car' to define a method: public Engine getEngine() { .. }

But what if you call **getEngine()** from another package? (see the next slide)

An example in Java

```
package qa;

public class Car {
  private Engine eng;

  public Car() {
     eng = new Engine(1600);
  }

  public Engine getEngine() {
    return eng;
  }
}
```

```
package qa;

class Engine {
  public Engine(int capacity) {
    // code
  }
}
```

```
package qa;

public static void main(String[] args) {
  Car myCar = new Car();
  Engine myEngin = myCar.getEngine();
}
```

qa package

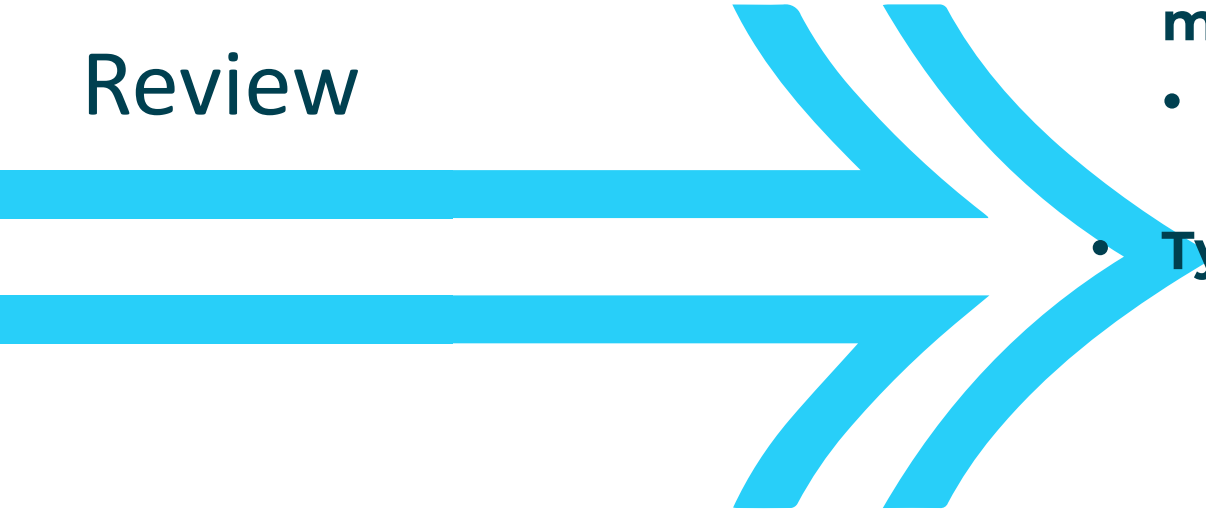
sales package

```
package sales;
import qa.*;

public class Program {
    public static void main(String[] args) {
        Car myCar = new Car();

            ✘ Engine myEngin = myCar.getEngine();
    }
}
```

Cannot hold an Engine reference here

Engine is not be visible in this package

Review



- **Types can have:**
  - Fields and Methods
  - Special constructor methods for object initialisation
  - Each can be instance (non-static) or static (belongs to class)

- **Member accessibility can be controlled with modifiers**
  - `public`, `protected`, `<default>` and `private` …
- **Type visibility though is `public` or `<default>`**

Hands on Lab

Author your own types – class Account

- Instantiation using multiple constructors
- Manipulating the multiple instances
- Using a combination of instance and shared data