POWERING POTENTIAL

Inheritance – Getting Started

CONTENTS

- **Objectives**
  - To add functionality to existing classes using inheritance
- **Contents**
  - Basic concepts of inheritance
  - Extending a simple class
- **Hands-on labs**

Base and derived classes

- **A class can inherit the features of another class**
  - The original class is the ‘super/base’ class
  - The new class is the ‘sub / derived’ class
- **The ‘sub’ class can:**
  - Utilise all the features of the super class
  - Override certain behaviour of the super class
  - Add new features
- **Inheritance is a fundamental object-oriented concept**

Existing code in the super class can be reused by the subclass

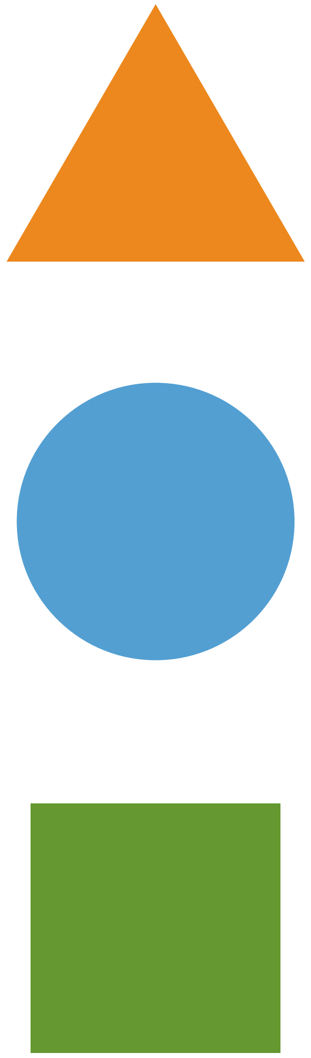
New classes can be defined simply in the terms of their differences from an existing class

Inheritance in action

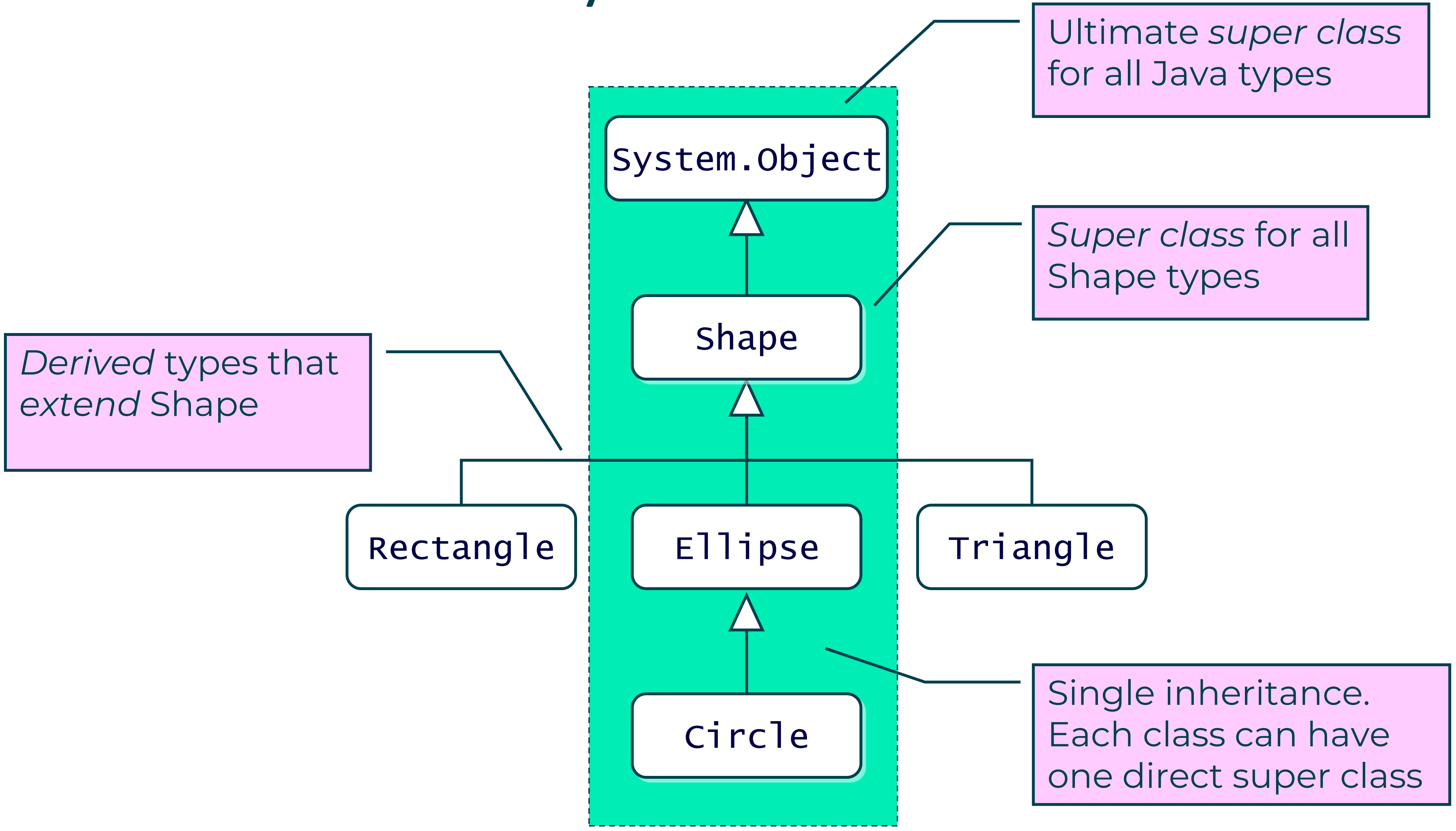
- **A vector graphics program**
  - Lots of commonality
    - `position` and `colour` fields
    - `draw` method
  - Want to benefit from re-use

- **Create a base class called Shape**
  - Implement common code there

- **Derive classes from `Shape`**
  - `Rectangle`, `Ellipse`, `Triangle`



The inheritance hierarchy


Ultimate *super class* for all Java types
System.Object
Super class for all Shape types
Shape
Derived types that *extend* Shape
Rectangle
Ellipse
Triangle
Circle
Single inheritance. Each class can have one direct super class

Specifying the base class

Super class the sub class extends

```
public class Shape {

  private Point position;
  private Color colour;
  ...
}
```

Sub classes extending the super class

```
public class Rectangle extends Shape {
   ...
}

public class Ellipse extends Shape {
   ...
}

public class Circle extends Ellipse {
   ...
}
```

Rectangle specific members

Ellipse specific members

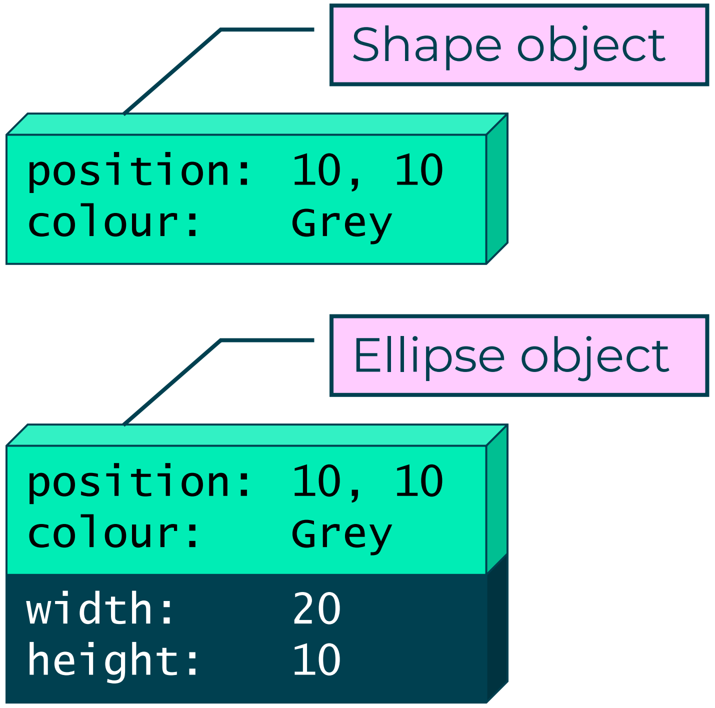
Circle specific members

Sub class inherits all the super class fields

```
public class Shape {
  private Point position;
  private Color colour;
   ...
}
```

Would be exposed via public get'ters

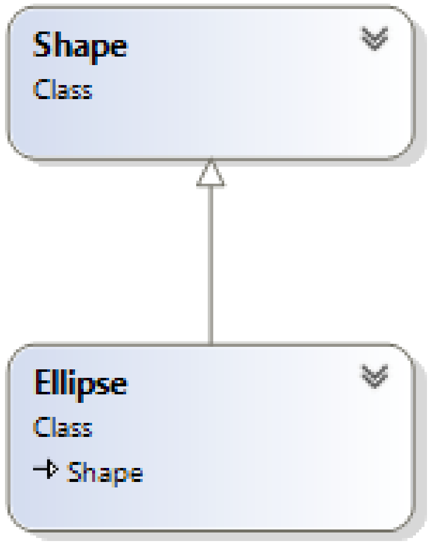
```
public class Ellipse extends Shape
{
  private int width;
  private int height;
   ...
}
```



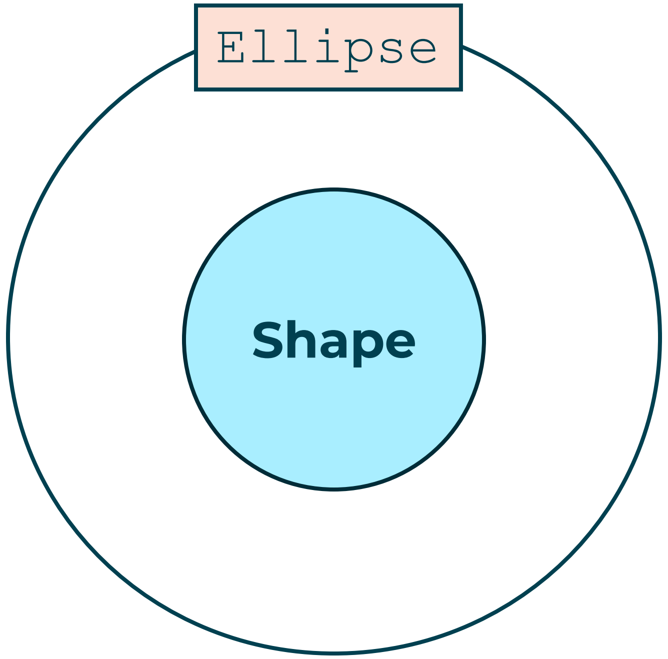
A sub type is a kind of super type

Constructing the derived objects

- **Base class constructors are not inherited**
  - But, default constructor of the base class is called
- **You can invoke base class constructor**
  - Mandatory if there is no default (no argument) constructor in the base class



Before constructing an Ellipse, system should construct the Shape object within



Derived class constructor

```
class Shape {
  private Point position;
  private Color colour;

  public Shape(Point pos, Color col) {
        position = pos;
        colour = col;
  }
}
```

No default .ctor
So all derived classes must invoke this .ctor

```
class Ellipse extends Shape {
  private int width, height;

  public Ellipse(Point position, int width, int height, Color colour) {
        super(position, colour);
        this.width = width;
        this.height = height;
  }
}
```

Calling base .ctor to initialise base fields

```
Ellipse e1 = new Ellipse(new Point(4,7), 23, 24, Color.RED);
```

Derived class constructor – chaining

```
class Ellipse extends Shape {
  private int width, height;

  public Ellipse(Point position, int width, int height, Color colour) {
        super(position, colour);
        this.width = width;
        this.height = height;
  }
  public Ellipse(Point pos) {
        this(pos, 10, 10, Color.BLUE);
  }
}
```

Calling base constructor initialise base fields

Calling Ellipse constructor

```
Ellipse e1 = new Ellipse(new Point(4,7));
```

Using Inheritance for creating custom exceptions

- **Custom exception class must derive from `Exception`**
  - Duplicate constructors
    - Pass ‘`String message`’ up to base class (the only time you can write to the inherited message field)
    - Can add additional methods
- **View code example on the next slide …**

Example

```java
public class QAException extends Exception {

    public QAException(String message) {
        super(message);
    }
    public QAException() {
        super("General error");
    }

    public void log() {
        // log the message field
    }
}
```

```java
void methodY() throws QAException {
    throw new QAException();
}
```

```java
void methodX() {
    try {
        methodY();
    } catch (QAException e) {
        System.out.println(e.getMessage());
        e.log();
    }
}
```

Review

- **Why do we do inheritance?**
  - Code reuse
  - Perhaps there will be other reasons soon!

- **Sub class inherits and can add additional functionality**

Hands-on labs

- Working with inheritance