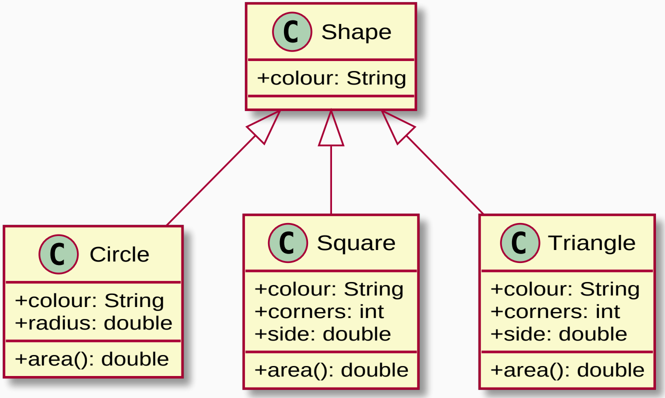
# Abstract Classes

COMP122: Object-Oriented Programming

Patrick Totzke

totzke@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL



C Shape

+colour: String

C Circle

+colour: String
+radius: double

+area(): double

C Square

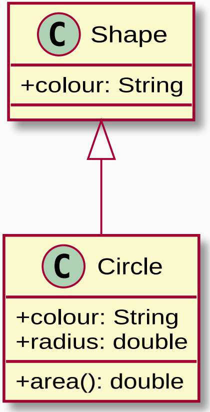
+colour: String
+corners: int
+side: double

+area(): double

C Triangle

+colour: String
+corners: int
+side: double

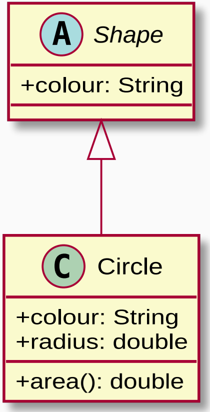
+area(): double



```java
public class Shape {
  public String colour;
}
```

```java
public class Circle extends Shape {
  public double radius;
  public double area(){
    return (radius*radius)*Math.PI;
  }
}
```

```java
Shape s = new Circle();
Shape s = new Shape();
```



```java
public abstract class Shape {
  public String colour;
}
```

```java
public class Circle extends Shape {
  public double radius;
  public double area(){
    return (radius*radius)*Math.PI;
  }
}
```

```java
Shape s = new Circle();
Shape s = new Shape();
```

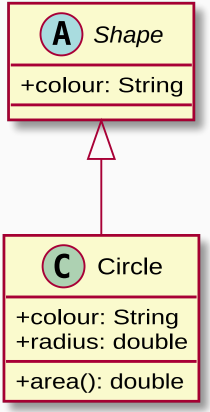
# Abstract

Abstract classes

- cannot be instantiated
- but they can be extended and (concrete) subclasses can be.

Abstract methods

- do not provide a (full) implementation.
- They have to be overrode by subclasses.



A *Shape*

+colour: String

C Circle

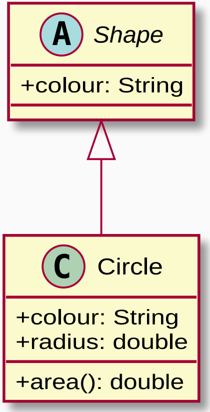
+colour: String
+radius: double

+area(): double

```
public abstract class Shape {
  public String colour;
}
```

```
public class Circle extends Shape {
  public double radius;
  public double area(){
    return (radius*radius)*Math.PI;
  }
}
```

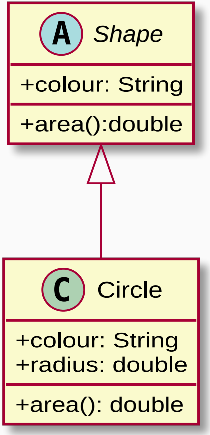
```
Shape s = new Circle();
double a = s.area();
```



```java
public abstract class Shape {
  public String colour;
}
```

```java
public class Circle extends Shape {
  public double radius;
  public double area(){
    return (radius*radius)*Math.PI;
  }
}
```

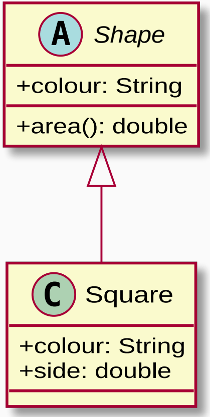
```java
Shape s = new Circle();
double a = s.area();
double a = ((Circle) s).area();
```



```java
public abstract class Shape {
  public String colour;
  public double area(){ ... }
}
```

```java
public class Circle extends Shape {
  public double radius;
  public double area(){
    return (radius*radius)*Math.PI;
  }
}
```

```java
Shape s = new Circle();
double a = s.area();
double a = ((Circle) s).area();
```



A Shape

+colour: String

+area(): double

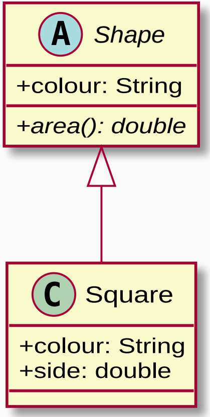
C Square

+colour: String
+side: double

```
public abstract class Shape {
  public String colour;
  public double area(){ ... }
}
```

```
public class Square extends Shape {
  public double side;
}
```

```
Shape s = new Square();
double a = s.area();
```



A *Shape*

+colour: String

*+area(): double*

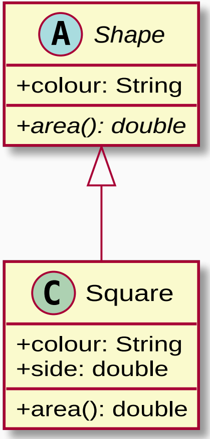
C Square

+colour: String
+side: double

```java
public abstract class Shape {
  public String colour;
  public abstract double area();
}
```

```java
public class Square extends Shape {
  public double side;
}
```

```java
Shape s = new Square();
double a = s.area();
```



A *Shape*

+colour: String

*+area(): double*

C Square

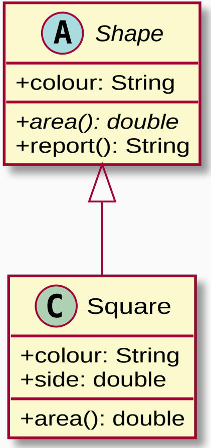
+colour: String
+side: double

+area(): double

```java
public abstract class Shape {
  public String colour;
  public abstract double area();
}
```

```java
public class Square extends Shape {
  public double side;
  public double area(){
    return side*side;
  }
}
```

```java
Shape s = new Square();
double a = s.area();
```



A *Shape*

+colour: String

*+area(): double*
+report(): String

C Square

+colour: String
+side: double

+area(): double

```java
public abstract class Shape {
  public String colour;
  public abstract double area();
  public String report() {
    return "My area is " + area();
}
```

```java
public class Square extends Shape {
  public double side;
  public double area(){
    return side*side;
  }
}
```

```java
Shape s = new Square();
String r = s.report();
```

# Abstract Classes

There are two good reasons for making a class `abstract`:

1. to prevent it from being instantiated
2. to enforce that concrete (instantiable) subclasses override some method.