SWEN20003

Workshop 7, Week 8

Eleanor McMurtry, University of Melbourne

UML

A structured way to diagram your designs

- **UML** (Unified Modelling Language) is a formal approach to software diagramming

A structured way to diagram your designs

- **UML** (Unified Modelling Language) is a formal approach to software diagramming
- UML helps us:
  - **understand** our design

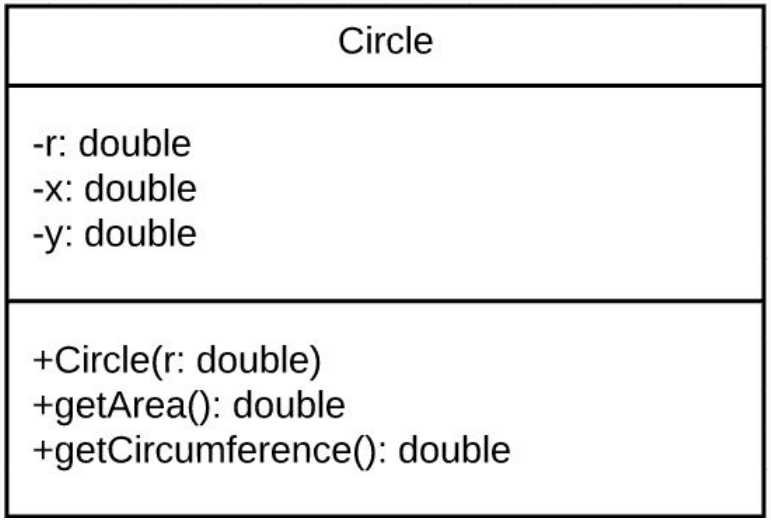
A structured way to diagram your designs

- **UML** (Unified Modelling Language) is a formal approach to software diagramming
- UML helps us:
  - **understand** our design
  - **share** our design

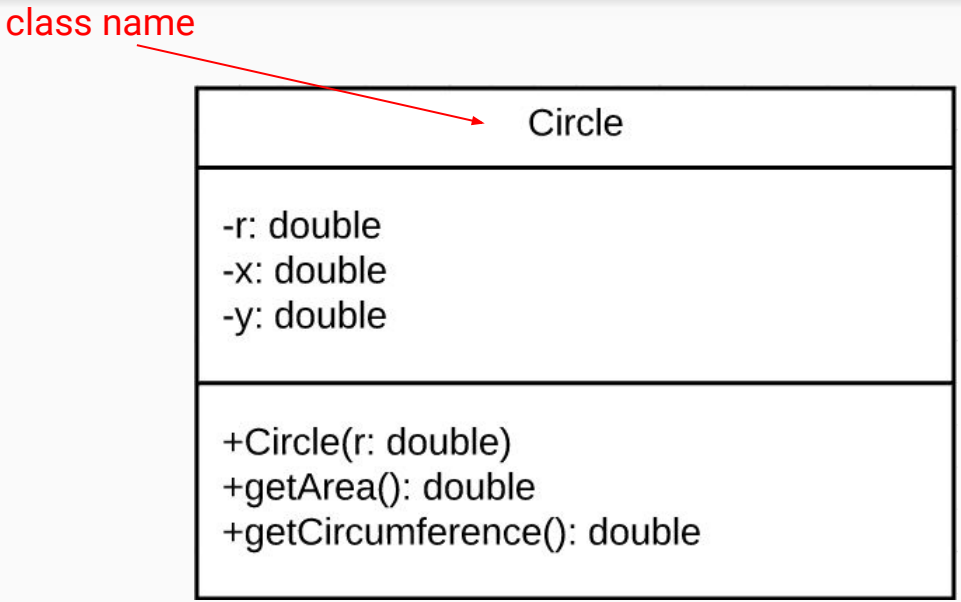
A structured way to diagram your designs

- **UML** (Unified Modelling Language) is a formal approach to software diagramming
- UML helps us:
  - **understand** our design
  - **share** our design
  - **review** our design

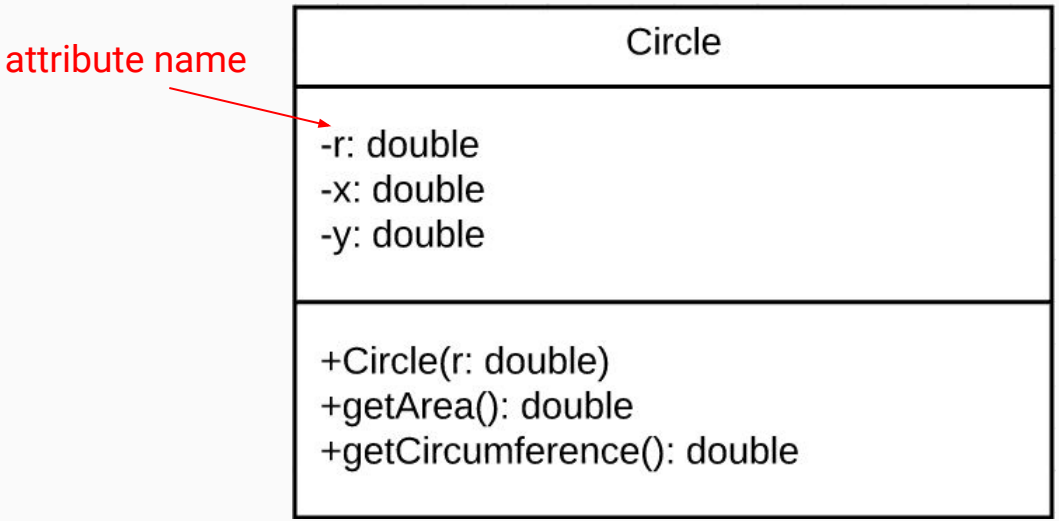
A class in UML



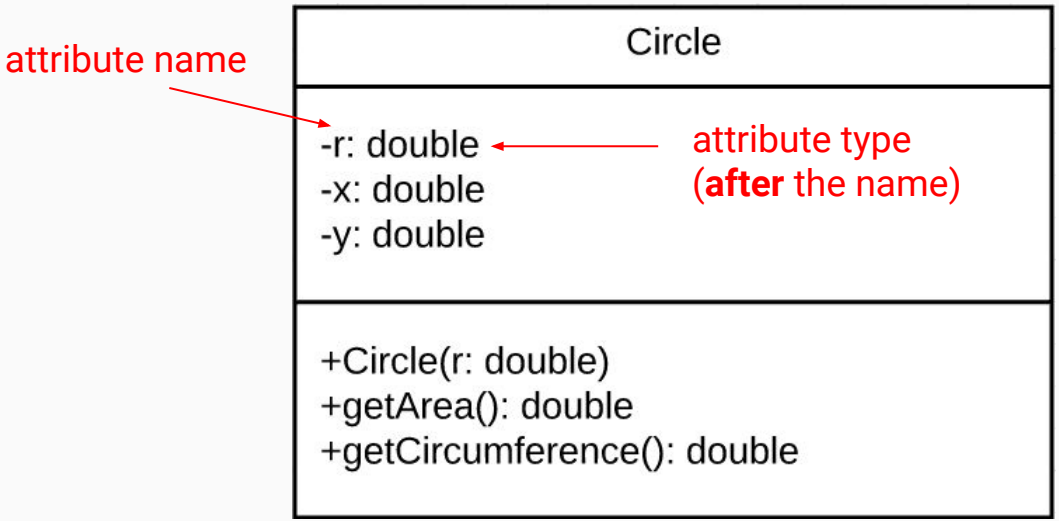
A class in UML



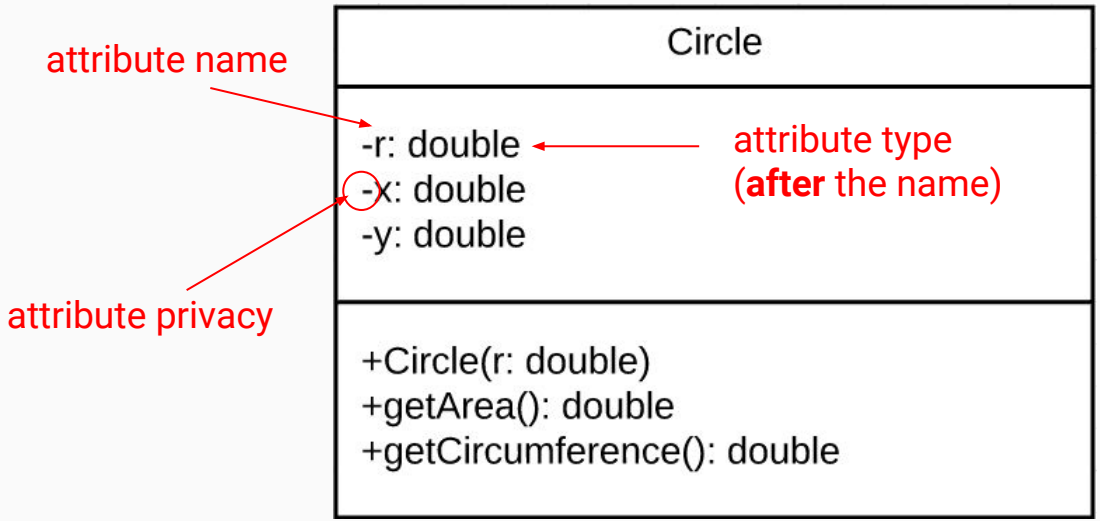
A class in UML



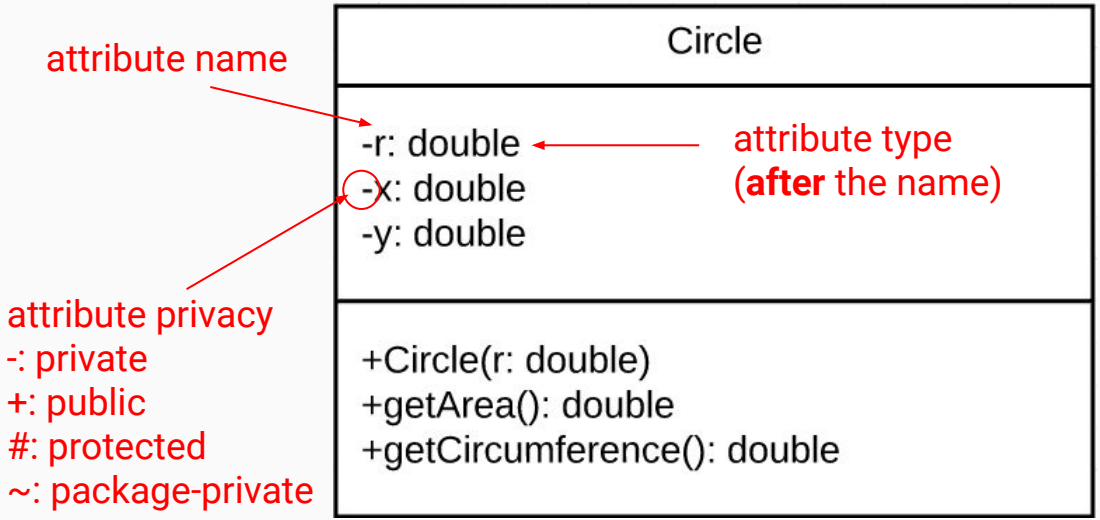
A class in UML



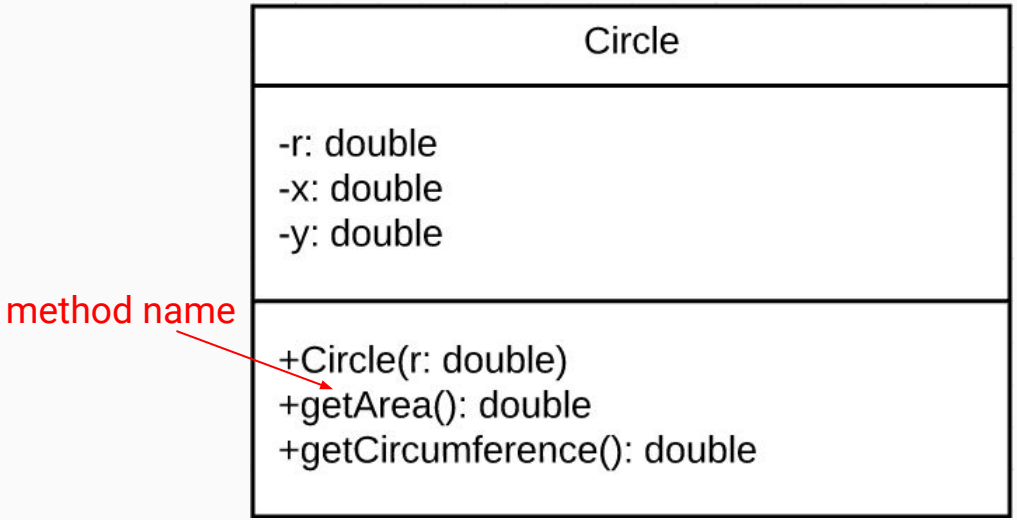
A class in UML



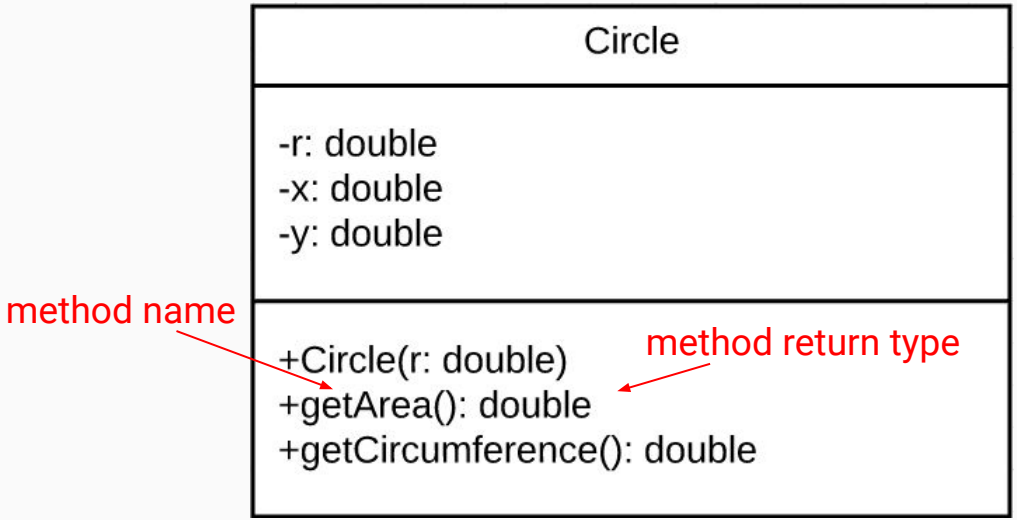
A class in UML



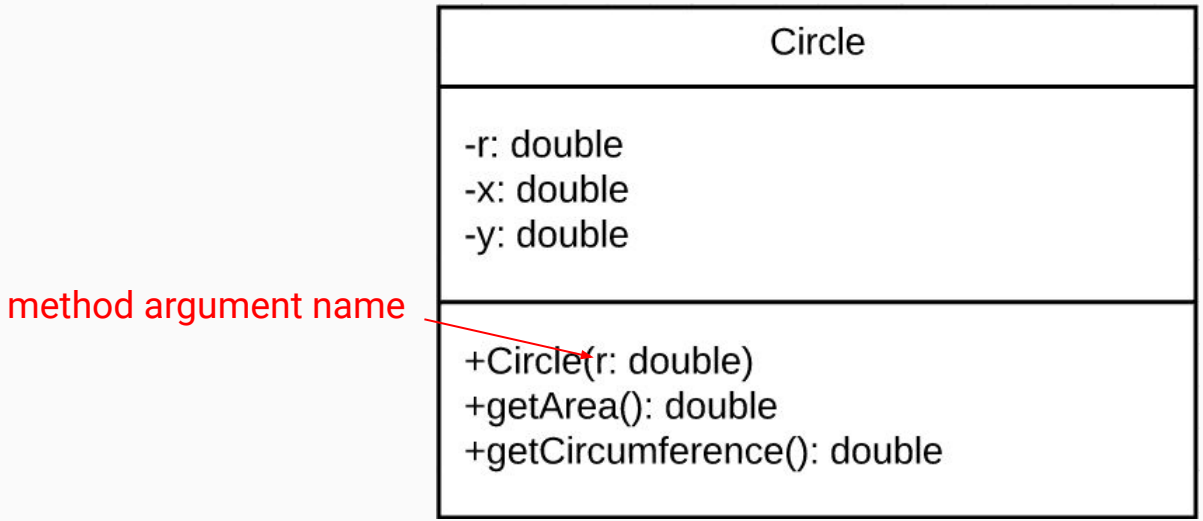
A class in UML



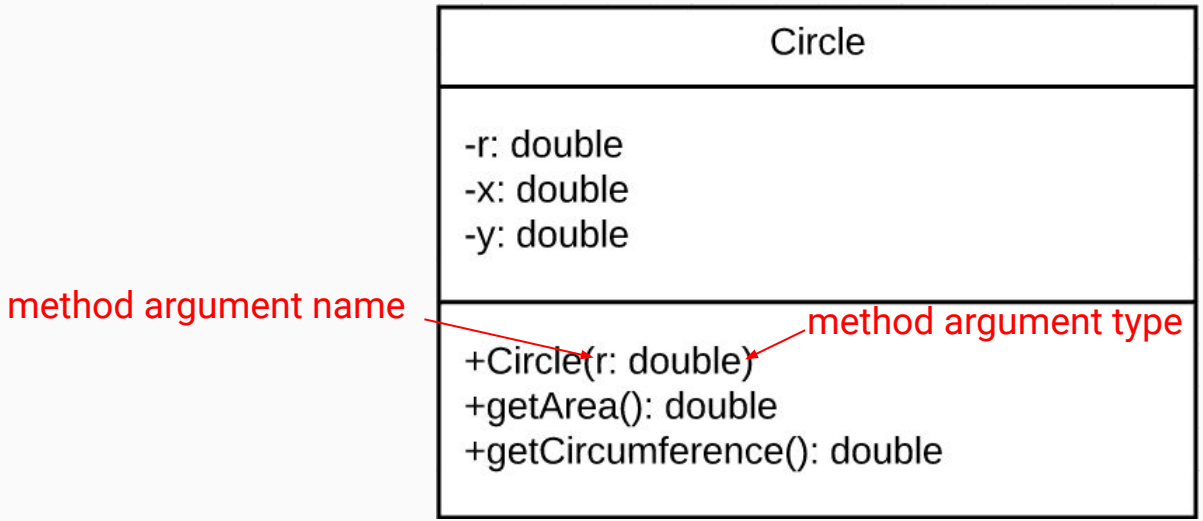
A class in UML



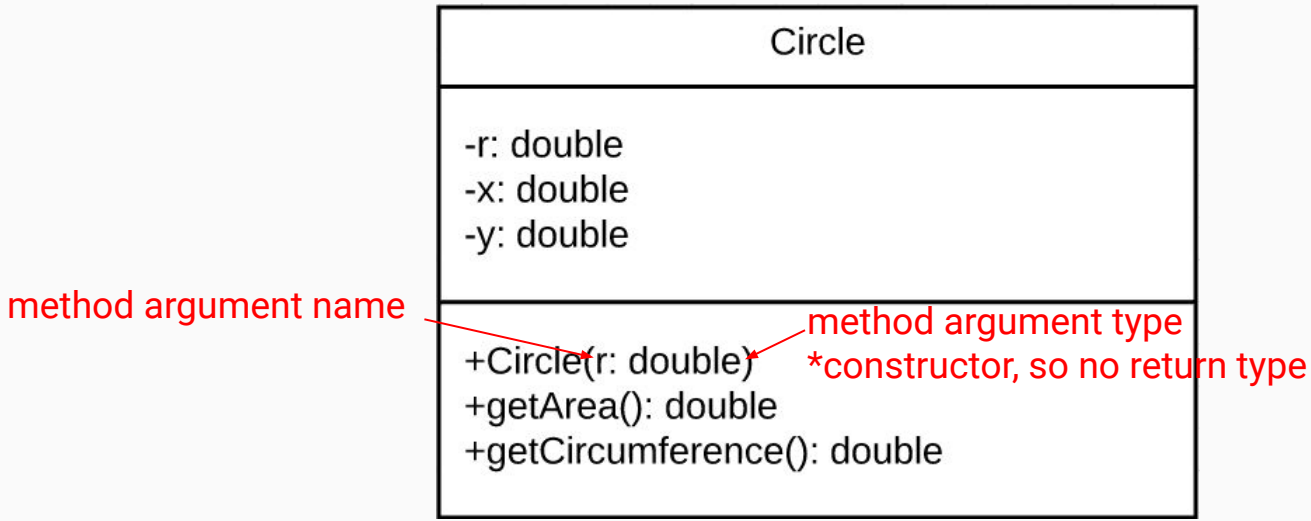
A class in UML



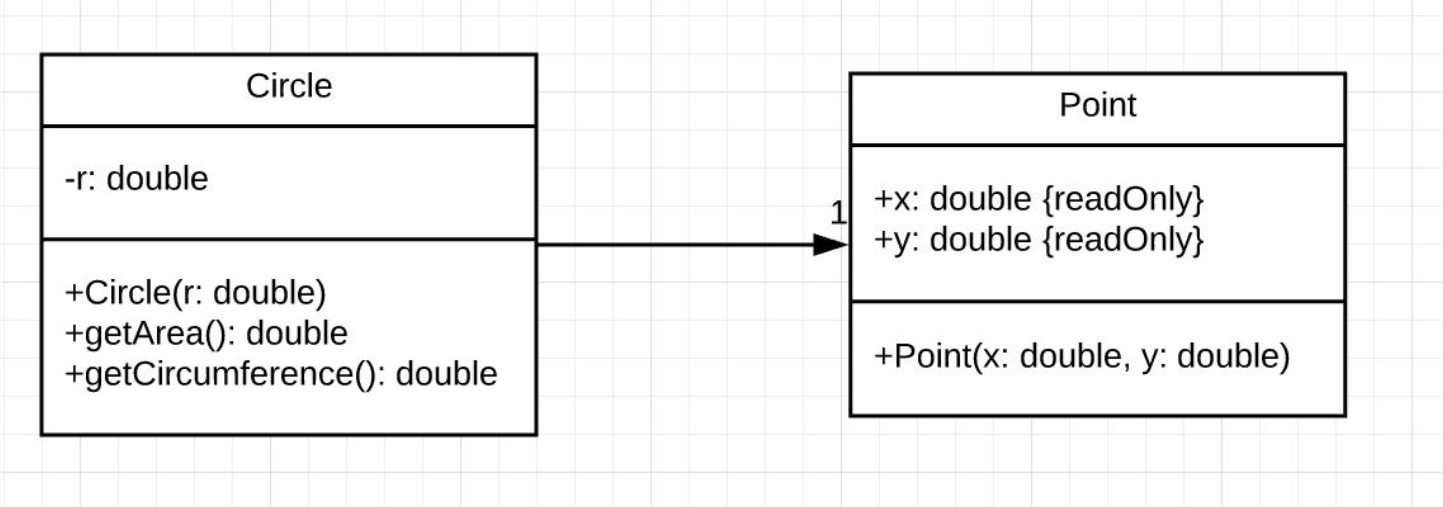
A class in UML



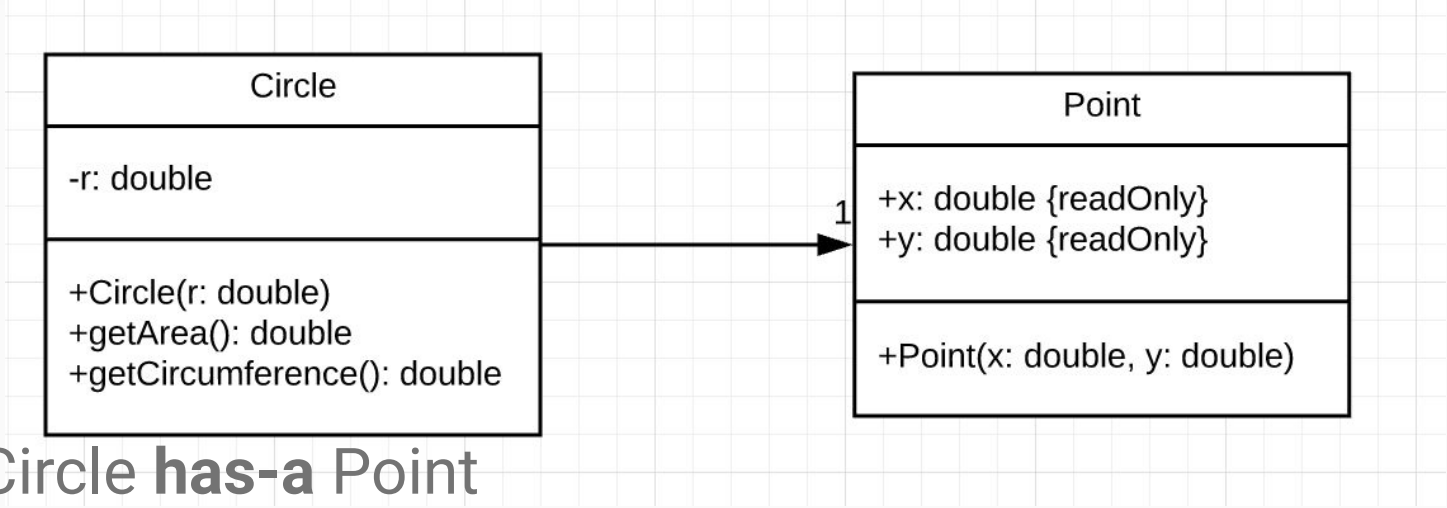
A class in UML



Relationships in UML: **Associations**

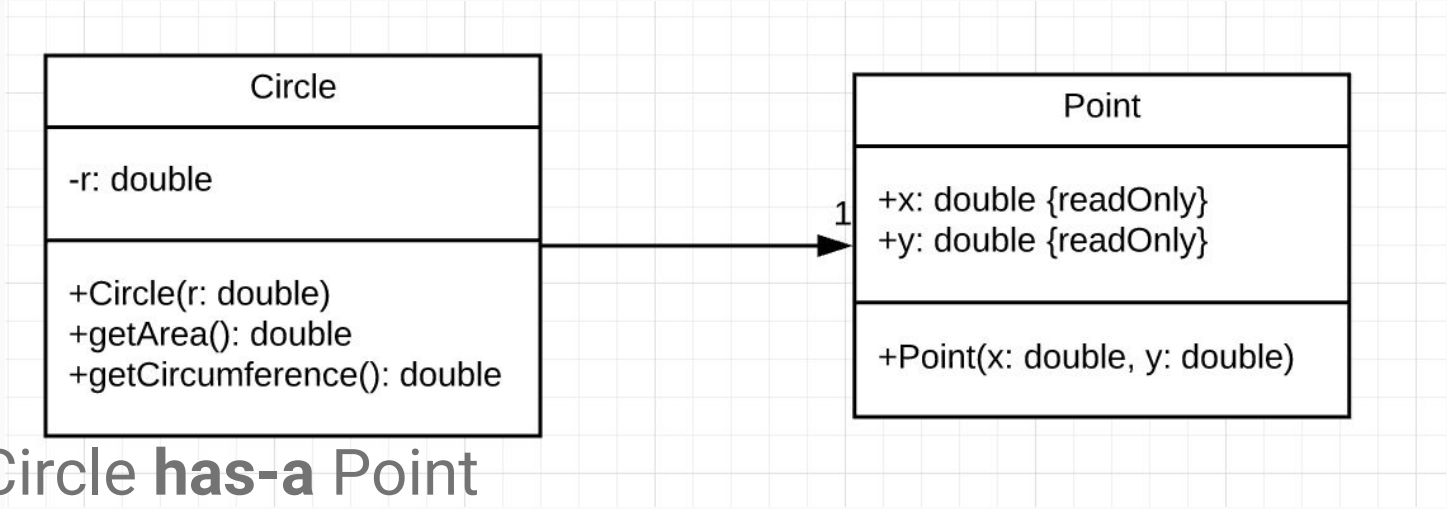


Relationships in UML: **Associations**



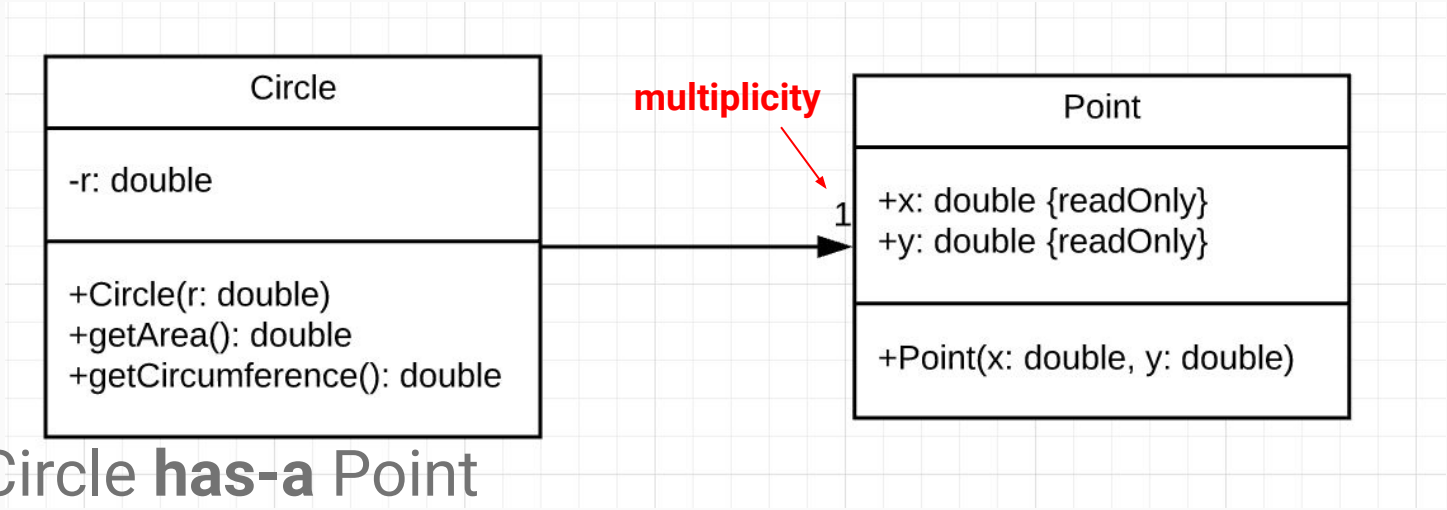
- Circle **has-a** Point

Relationships in UML: **Associations**



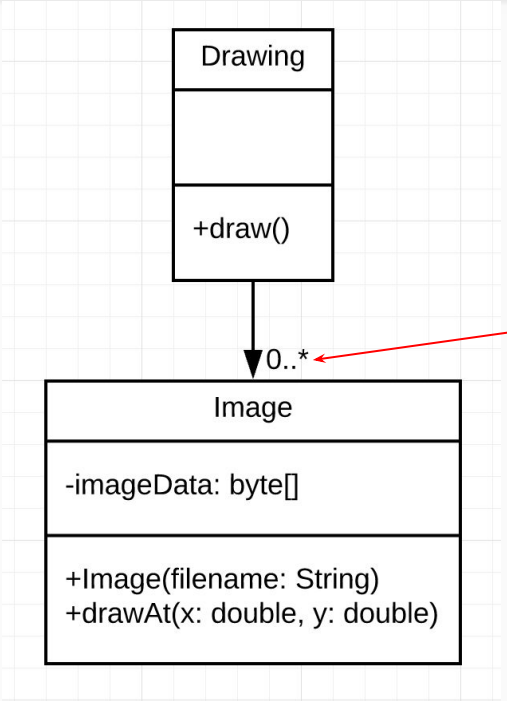
- Circle **has-a** Point
- Use **instead of** attributes for instances of classes on diagram

Relationships in UML: **Associations**



- Circle **has-a** Point
- Use **instead of** attributes for instances of classes on diagram

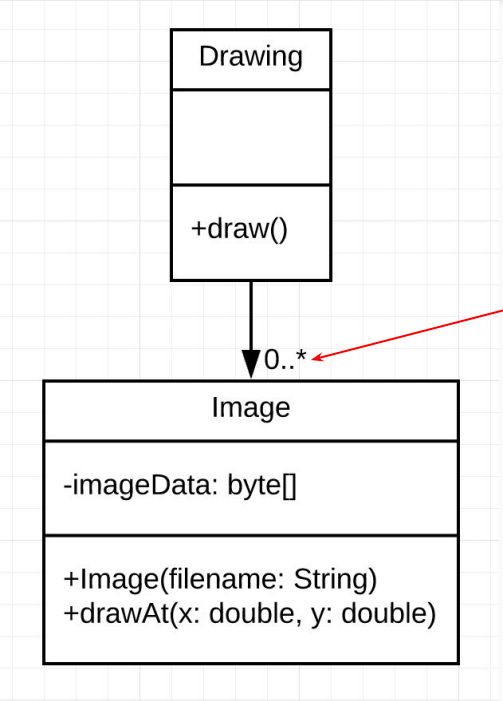
Relationships in UML: **Multiplicity**




```java
public class Drawing {
    private Image[] images = new Image[10];

    public void draw() {
        for (Image image : images) {
            image.drawAt(0, 0);
        }
    }
}
```

Relationships in UML: **Multiplicity**



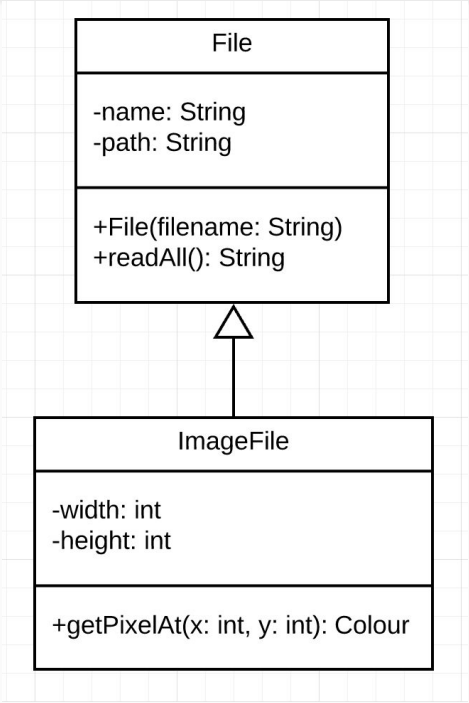

```java
public class Drawing {
    private List<Image> images = new ArrayList<>();

    public void draw() {
        for (Image image : images) {
            image.drawAt(0, 0);
        }
    }
}
```

In “real” Java, we usually avoid arrays.

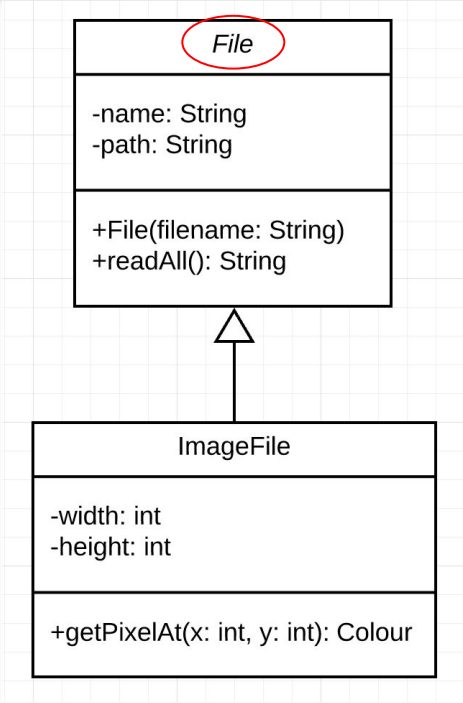
Relationships in UML: **Generalisation**

- Arrow points **towards the parent** class

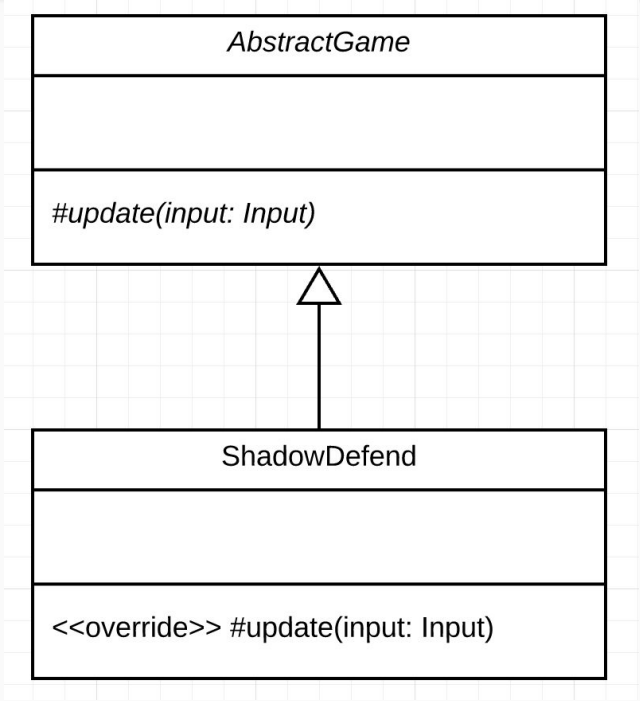


Relationships in UML: **Generalisation**

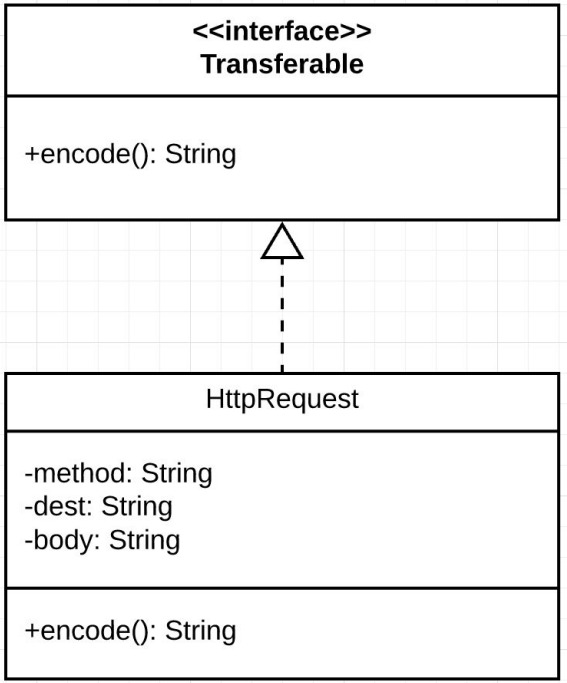
- Arrow points **towards the parent** class
- Italics = **abstract**



Relationships in UML: **Overriding**



Relationships in UML: **Realisation**



UML tools

- **LucidChart:** https://lucidchart.com/ (my recommendation)
- **diagrams.net:** https://app.diagrams.net/
- **StarUML (offline):** http://staruml.io/

Generics

Type parameters

- **Generics** allow us to write classes that work with **any** (non-primitive) type.

```
public class Pair<T, U> {
    public final T fst;
    public final U snd;

    public Pair(T fst, U snd) {
        this.fst = fst;
        this.snd = snd;
    }
}
```

Type parameters

- **Generics** allow us to write classes that work with **any** (non-primitive) type.

```java
public class Pair<T, U> {
    public final T fst;
    public final U snd;

    public Pair(T fst, U snd) {
        this.fst = fst;
        this.snd = snd;
    }
}
```

```java
class Program {
    public static void main(String[] args) {
        Pair<Double, Double> coord = new Pair<>(32.0, 100.0);
        Pair<String, Integer> student = new Pair<>("Eleanor", 700000);
    }
}
```

Type parameters

- Generic classes can do **compile-time** type checking to prevent incorrect code.

```
class Program {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>();
        names.add(5);
        double x = names.get(0);
    }
}
```

Limitations of generics

- Generic parameters (e.g. T, U) cannot be used
  - to create instances: `T foo = new T();`
  - with instanceof: `if (foo instanceof T) {`

Conditional generics

- You can restrict the types that may be used for a generic parameter.

```
public class NumberPair<T extends Number, U extends Number> {
    public final T fst;
    public final U snd;

    public NumberPair(T fst, U snd) {
        this.fst = fst;
        this.snd = snd;
    }
}
```

Try out the workshop questions.