

2. Class, Objects & Methods

- A **Class** can be considered as a blueprint using which you can create as many objects as you like.
- **Objects** have state and behaviour
- **Abstraction** is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user.
- **Encapsulation** simply means binding object state(fields) and behaviour (methods) together. If you are creating class, you are doing encapsulation.
- **Association** establishes relationships between two Objects so as to enable Method Invocation

Class Specification

```
public class Charge
{
    private final double rx, ry;
    private final double q;

    public Charge(double x0, double y0, double q0)
    { rx = x0; ry = y0; q = q0; }

    public double potentialAt(double x, double y)
    {
        double k = 8.99e09;
        double dx = x - rx;
        double dy = y - ry;
        return k * q / Math.sqrt(dx*dx + dy*dy);
    }

    public String toString()
    { return q + " at " + "(" + rx + ", " + ry + ")"; }

    public static void main(String[] args)
    {
        double x = Double.parseDouble(args[0]);
        double y = Double.parseDouble(args[1]);
        Charge c1 = new Charge(0.51, 0.63, 21.3);
        Charge c2 = new Charge(0.13, 0.94, 81.9);
        double v1 = c1.potentialAt(x, y);
        double v2 = c2.potentialAt(x, y);
        StdOut.printf("%.2e\n", (v1 + v2));
    }
}
```

Diagram annotations:

- class name**: points to `Charge` in `public class Charge`
- instance variables**: points to `private final double rx, ry;` and `private final double q;`
- constructor**: points to `public Charge(double x0, double y0, double q0)`
- instance methods**: points to `potentialAt` and `toString`
- test client**: points to `main`
- create and initialize object**: points to `new Charge(0.51, 0.63, 21.3);` and `new Charge(0.13, 0.94, 81.9);`
- object name**: points to `c1` and `c2`
- invoke constructor**: points to `new Charge(0.51, 0.63, 21.3);` and `new Charge(0.13, 0.94, 81.9);`
- instance variable names**: points to `rx` and `ry` in `potentialAt`
- invoke method**: points to `c1.potentialAt(x, y)` and `c2.potentialAt(x, y)`