# Mocks

**1**. Assuming `InvoiceFilter` is a business entity and `IssuedInvoices` is the DAO (data access object) connecting to a database, how would you go testing them (unit tests, mocks, integration tests, etc)?

```
public class InvoiceFilter {
    private IssuedInvoices invoices;

    public InvoiceFilter(IssuedInvoices invoices) {
      this.invoices = invoices;
    }
    public List<Invoice> filter() {
      return invoices.all().stream()
          .filter(invoice -> invoice.getValue() < 100.0)
          .collect(toList());
    }
}
```

**2**. Consider the following class:

```
public class Geometry {
   private Math math;

   public Geometry(Math math) {
       this.math = math;
   }

   public double distance(Point p1, Point p2) {
       return math.sqrt(
           math.pow(p2.getX() - p1.getX(), 2) + math.pow(p2.getY() - p1.getY(), 2));
   }
}
```

Should you mock the Math class (see the following page)?

```java
// Example test without mocking
@Test
public void testDistance1() {
    var geometry = new Geometry(new Math());

    Point p1 = new Point(0, 0);
    Point p2 = new Point(3, 4);

    assertEquals(5, geometry.distance(p1, p2));
}
// Example test with mocking
@Test
public void testDistance2() {
    var mathMock = Mockito.mock(Math.class);
    when(mathMock.pow(3)).thenReturn(109);
    when(mathMock.pow(4)).thenReturn(116);
    when(mathMock.sqrt(225)).thenReturn(5);

    var geometry = new Geometry(mathMock);
    Point p1 = new Point(0, 0);
    Point p2 = new Point(3, 4);

    assertEquals(5, geometry.distance(p1, p2));

    verify(mathMock).pow(3);
    verify(mathMock).pow(4);
}
```

**3**. Come up with as many advantages and disadvantages of mocking (stubs and spies including) as you can.