SAT: Test doubles and Mocks

1. Mocks, stubs, and fakes. What are they, and how do they differ from each other?

Fakes have real, working implementations of the class they simulate. However, they usually do the same task in a much simpler way. Stubs provide hard-coded answers to the calls that are performed during the test. Unlike fakes, stubs do not have a working implementation. Mock objects act like stubs but are preconfigured to know what kind of interactions should occur with them.

2. What are advantages and disadvantages of mocking?

```
Advantages:
- More control
- Faster simulations
- Reflect on dependencies and class interactions
Disadvatages:
- Test the mock not the code
- Higher coupling between the tests and the code they test
```

3. According to the guidelines provided in the book, what types of classes should you mock, and which should you not mock?

Anything that is infrastructure-related, too complex, or too slow is a good candidate to be replaced by mocks.
On the other hand, entity classes, simple utility functions, and data holders are not commonly mocked.

4. You are testing a system that triggers advanced events based on complex combinations of external, boolean conditions relating to the weather (outside temperature, amount of rain, wind, and so on). The system has been designed cleanly and consists of a set of cooperating classes, each of which has a single responsibility. Explain how would you use mocking in your testing?

You use mocks to control the external conditions and to observe the event being triggered.

5. Suppose your aim is to test `applyDiscount` in the following class:

```
public class ChristmasDiscount {

    public double applyDiscount(double rawAmount) {
        LocalDate today = LocalDate.now();

        double discountPercentage = 0;
        boolean isChristmas = today.getMonth()== Month.DECEMBER
                && today.getDayOfMonth()==25;

        if(isChristmas)
            discountPercentage = 0.15;

        return rawAmount - (rawAmount * discountPercentage);
    }
}
```

Note that `LocalDate.now()` is a static method of Java's Time API. How would you handle the dependency on this method in your testing?

```
public class Clock {
    public LocalDate now() {
        return LocalDate.now();
    }
    // any other date and time operation here...
}
private final Clock clock = mock(Clock.class);
private final ChristmasDiscount cd = new ChristmasDiscount(clock);
@Test
public void christmas() {
    LocalDate christmas = LocalDate.of(2015, Month.DECEMBER, 25);
    when(clock.now()).thenReturn(christmas);

    double finalValue = cd.applyDiscount(100.0);
    assertThat(finalValue).isCloseTo(85.0, offset(0.001));
}
@Test
public void notChristmas() {
    LocalDate notChristmas = LocalDate.of(2015, Month.DECEMBER, 26);
    when(clock.now()).thenReturn(notChristmas);

    double finalValue = cd.applyDiscount(100.0);
    assertThat(finalValue).isCloseTo(100.0, offset(0.001));
}
```