Event.java

```
import java.time.LocalDateTime;

public class Event {
  LocalDateTime start;
  LocalDateTime end;
  String location;

  public Event(LocalDateTime start, LocalDateTime end, String location) {
    this.start = start;
    this.end = end;
    this.location = location;
  }

  /*
    @param other the Event to compare to
    @return true if the other event happens at an overlapping time and the same location
  */
  public boolean conflict(Event other) {









  }
}
```

EventTest.java

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
import java.time.LocalDateTime;

public class EventTest {









  @Test
  public void testConflict() {

















  }
}
```

```
$ javac -cp hamcrest-core-1.3.jar:junit-4.12.jar:. EventTest.java
$ java -cp hamcrest-core-1.3.jar:junit-4.12.jar:. org.junit.runner.JUnitCore EventTest
JUnit version 4.12
.
Time: 0.012

OK (1 test)
```

Class LocalDateTime (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/LocalDateTime.html)

public static LocalDateTime of(int year, int month, int dayOfMonth, int hour, int minute)

Obtains an instance of LocalDateTime from year, month, day, hour and minute, setting the second and nanosecond to zero.

public int compareTo(**LocalDateTime** other)

Compares this date-time to another date-time. Returns a negative number if this date-time happened earlier, 0 if they are the same and a positive number otherwise. (See java.lang.Comparable)

Class org.junit.Assert (https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html)

public static void assertEquals(**Object** expected, **Object** actual)

Asserts that two objects are equal. Uses the .equals() method to compare the objects.

public static void assertEquals(**String** message, **Object** expected, **Object** actual)

Asserts that two objects are equal. Uses the .equals() method to compare the objects. Uses the given message as part of the failure description.

Annotation Test (https://junit.org/junit4/javadoc/4.12/org/junit/Test.html)

The Test annotation tells JUnit that the **public void** method to which it is attached can be run as a test case. To run the method, JUnit first constructs a fresh instance of the class then invokes the annotated method. Any exceptions thrown by the test will be reported by JUnit as a failure. If no exceptions are thrown, the test is assumed to have succeeded.