Ainda temos muito código não testado no nosso sistema! Veja a classe `Leilao`, por exemplo. Ela não tem teste nenhum!

```
public class Leilao {

    private String descricao;
    private List<Lance> lances;

    public Leilao(String descricao) {
        this.descricao = descricao;
        this.lances = new ArrayList<Lance>();
    }

    public void propoe(Lance lance) {
        lances.add(lance);
    }


    public String getDescricao() {
        return descricao;
    }


    public List<Lance> getLances() {
        return Collections.unmodifiableList(lances);
    }
}
```

Apesar de ser simples, essa classe é de extrema importância para o funcionamento do nosso sistema e, portanto, merece ser testada. O método `propoe()`, em especial, é essencial para o leilão e provavelmente sofrerá mudanças no decorrer do projeto. Por isso, ela precisa ser testada para termos certeza que ela esteja funcionando conforme se espera. Inicialmente, vamos analisar se um determinado lance que foi proposto ficará armazenado no Leilão. Para isso, temos dois casos a serem averiguados: a realização de apenas um lance e a de mais de um lance.

O código para realizar tal teste não tem muito segredo. Começaremos instanciando um leilão e serão propostos alguns lances nele:

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class LeilaoTest {

    @Test
    public void deveReceberUmLance() {
        Leilao leilao = new Leilao("Macbook Pro 15");
        assertEquals(0, leilao.getLances().size());

        leilao.propoe(new Lance(new Usuario("Steve Jobs"), 2000));

        assertEquals(1, leilao.getLances().size());
        assertEquals(2000, leilao.getLances().get(0).getValor(), 0.00001)
    }

    @Test
    public void deveReceberVariosLances() {
        Leilao leilao = new Leilao("Macbook Pro 15");
        leilao.propoe(new Lance(new Usuario("Steve Jobs"), 2000));
        leilao.propoe(new Lance(new Usuario("Steve Wozniak"), 3000));

        assertEquals(2, leilao.getLances().size());
        assertEquals(2000, leilao.getLances().get(0).getValor(), 0.00001)
        assertEquals(3000, leilao.getLances().get(1).getValor(), 0.00001)
    }
}
```

Agora implementaremos duas novas regras de negócio no processo de lances em um leilão: - Uma pessoa não pode propor dois lances em sequência; - Uma pessoa não pode dar mais do que cinco lances no mesmo leilão.

Contudo, dessa vez, vamos fazer diferente. Estamos muito acostumados a implementar o código de produção e testá-lo ao final. Mas será que essa é a única maneira de desenvolver um projeto? Vamos tentar inverter e começar pelos testes! Além disso, vamos tentar ao máximo ser o mais simples possível, ou seja, vamos pensar no cenário mais simples naquele momento e implementar sempre o código mais simples que resolva o problema.

Vamos começar pelo cenário mais simples. Um novo leilão cujo mesmo usuário dê dois lances seguidos e, por isso, o último lance deve ser ignorado:

```
    @Test
    public void naoDeveAceitarDoisLancesSeguidosDoMesmoUsuario() {
        Leilao leilao = new Leilao("Macbook Pro 15");
        Usuario steveJobs = new Usuario("Steve Jobs");

        leilao.propoe(new Lance(steveJobs, 2000));
        leilao.propoe(new Lance(steveJobs, 3000));

        assertEquals(1, leilao.getLances().size());
        assertEquals(2000, leilao.getLances().get(0).getValor(), 0.00001)
    }
```

Ótimo, teste escrito. Ao rodar o teste, ele falha. Mas tudo bem já estávamos esperando por isso! Precisamos fazê-lo passar agora, da maneira mais simples possível. Vamos modificar o método `propoe()`. Agora ele, antes de adicionar o lance, verificará se o último lance da lista não pertence ao usuário que está dando o lance naquele momento:

```
    public void propoe(Lance lance) {
        if (lances.isEmpty() ||
                !lances.get(lances.size() - 1).getUsuario().equals(lance.ge
            lances.add(lance);
        }
    }
```

Veja que fazemos uso do método `equals()` na classe `Usuario`, e vamos precisar implementá-lo também, usando o atalho do Eclipse (Generate HashCode and Equals):

```
public class Usuario {
  // codigo anterior aqui

  @Override
  public boolean equals(Object obj) {
      if (this == obj)
          return true;
      if (obj == null)
          return false;
      if (getClass() != obj.getClass())
          return false;
      Usuario other = (Usuario) obj;
      if (id != other.id)
          return false;
      if (nome == null) {
          if (other.nome != null)
              return false;
      } else if (!nome.equals(other.nome))
          return false;
      return true;
  }
}
```

Se rodarmos o teste agora, ele passa! Nosso código funciona! Mas veja que o código que produzimos não está muito claro. O `if` em particular está confuso. Agora é uma boa hora para melhorar isso, afinal temos certeza que o código atual funciona! Ou seja, se melhorarmos o código e rodarmos o teste novamente, ele deverá continuar verde.

Vamos, por exemplo, extrair o código responsável por pegar o último elemento da lista em um método privado:

```
    public void propoe(Lance lance) {
        if (lances.isEmpty() || !ultimoLanceDado()
                .getUsuario().equals(lance.getUsuario())) {
            lances.add(lance);
        }
    }

    private Lance ultimoLanceDado() {
        return lances.get(lances.size() - 1);
    }
```

Perfeito. Vamos para a próxima regra de negócio: um usuário só pode dar no máximo 5 lances para um mesmo leilão. De novo, vamos começar pelo teste. Vamos criar um leilão e fazer um usuário dar 5 lances nele. Reparе que, devido a regra anterior, precisaremos intercalar os lances, já que o mesmo usuário não pode fazer dois lances em sequência:

```
    @Test
    public void naoDeveAceitarMaisDoQue5LancesDeUmMesmoUsuario() {
        Leilao leilao = new Leilao("Macbook Pro 15");
        Usuario steveJobs = new Usuario("Steve Jobs");
        Usuario billGates = new Usuario("Bill Gates");

        leilao.propoe(new Lance(steveJobs, 2000));
        leilao.propoe(new Lance(billGates, 3000));
        leilao.propoe(new Lance(steveJobs, 4000));
        leilao.propoe(new Lance(billGates, 5000));
        leilao.propoe(new Lance(steveJobs, 6000));
        leilao.propoe(new Lance(billGates, 7000));
        leilao.propoe(new Lance(steveJobs, 8000));
        leilao.propoe(new Lance(billGates, 9000));
        leilao.propoe(new Lance(steveJobs, 10000));
        leilao.propoe(new Lance(billGates, 11000));

        // deve ser ignorado
        leilao.propoe(new Lance(steveJobs, 12000));

        assertEquals(10, leilao.getLances().size());
        int ultimo = leilao.getLances().size() - 1;
        Lance ultimoLance = leilao.getLances().get(ultimo);
        assertEquals(11000.0, ultimoLance.getValor(), 0.00001);
    }
```

Ao rodarmos, o teste falhará! Excelente, vamos agora fazê-lo passar escrevendo o código mais simples:

```
    public void propoe(Lance lance) {

        int total = 0;
        for(Lance l : lances) {
            if(l.getUsuario().equals(lance.getUsuario())) total++;
        }

        if(lances.isEmpty() ||
                (!ultimoLanceDado().getUsuario().equals(lance.getUsuario())
                        && total < 5)) {
            lances.add(lance);
        }
    }
```

Pronto! O teste passa! Mas esse código está claro o suficiente? Podemos melhorar! De novo, uma ótima hora para refatorarmos nosso código. Vamos extrair a lógica de contar o número de lances de um usuário em um método privado:

```
    public void propoe(Lance lance) {
        if (lances.isEmpty() || !
            ultimoLanceDado().getUsuario().equals(lance.getUsuario()) &&
        qtdDelancesDo(lance.getUsuario()) < 5)) {
            lances.add(lance);
        }
    }

    private int qtdDelancesDo(Usuario usuario) {
        int total = 0;
        for(Lance lance : lances) {
            if(lance.getUsuario().equals(usuario)) total++;
        }
        return total;
    }
```

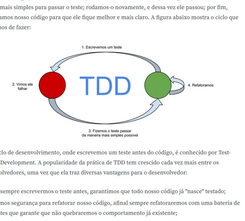
Rodamos o teste, e tudo continua passando. Podemos melhorar ainda mais o código, então vamos continuar refatorando. Dessa vez, vamos extrair aquela segunda condição do `if` para que a leitura fique mais clara:

```
    public void propoe(Lance lance) {
        if (lances.isEmpty() || podeDarLance(lance.getUsuario())) {
            lances.add(lance);
        }
    }

    private boolean podeDarLance(Usuario usuario) {
        return !ultimoLanceDado().getUsuario().equals(usuario)
            && qtdDelancesDo(usuario) < 5;
    }
```

Pronto! Os testes continuam passando! Poderíamos continuar escrevendo testes antes do código, mas vamos agora pensar um pouco sobre o que acabamos de fazer.

Em primeiro lugar, escrevemos um teste; o rodamos e o vimos falhar; em seguida, escrevemos o código mais simples para passar o teste; rodamos-o novamente, e dessa vez ele passou; por fim, refatoramos nosso código para que ele fique melhor e mais claro. A figura abaixo mostra o ciclo que acabamos de fazer:



Esse ciclo de desenvolvimento, onde escrevemos um teste antes do código, é conhecido por Test-Driven Development. A popularidade da prática de TDD tem crescido cada vez mais entre os desenvolvedores, uma vez que ela traz diversas vantagens para o desenvolvedor:

- Se sempre escrevermos o teste antes, garantimos que todo nosso código já "nasce" testado;
- Temos segurança para refatorar nosso código, afinal sempre refatoraremos com uma bateria de testes que garante que não quebraremos o comportamento já existente;
- Como o teste é a primeira classe que usa o seu código, você naturalmente tende a escrever código mais fácil de ser usado e, por consequência, mais fácil de ser mantido.

Vamos praticar!