Alunos: Arthur Vinicius Carboni Linzing, Mateus de Oliveira Lopes e Yan Gabriel Reis

RolagemDadosTest.java

```
package com.trabalhojava.sistemarpg.main;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class RolagemDadosTest {

	@Test
	void testeAtributo_deveEstarNoIntervalo() {
	RolagemDados dados = new RolagemDados();
	int valor = dados.testeAtributo(5);
	assertTrue(valor >= 6 && valor <= 25, "O valor da rolagem deve estar entre 6 e 25");
	}

	@Test
	void rodarAtributos_deveEstarNoIntervalo() {
	RolagemDados dados = new RolagemDados();
	int valor = dados.rodarAtributos();
	assertTrue(valor >= 3 && valor <= 18, "O valor do atributo deve estar entre 3 e 18");

	}
}
```

RacaDBDAOTest.java

```
package com.trabalhojava.sistemarpg.dao;

import com.trabalhojava.sistemarpg.model.Raca;
import com.trabalhojava.sistemarpg.model.Sistema;
import org.junit.jupiter.api.Test;

import java.sql.SQLException;

import static org.junit.jupiter.api.Assertions.*;

class RacaDBDAOTest {

	@Test
	void buscarPorCodigo_deveBuscarCorretamente() throws SQLException {
	Sistema sistema = new Sistema(1, "Tormenta");
	Raca raca = new Raca(15, "Teste", "D", 0,0,0,0,0,0,sistema);
	RacaDBDAO racaDB = new RacaDBDAO();
```

```
        racaDB.remover(raca);
        racaDB.insere(raca);
        Raca racaBusca = racaDB.buscarPorCodigo(15);
        int codigo = racaBusca.getRacaId();
        assertEquals(15, codigo, "O código da raca registrada deve ser 15");
        }

        @Test
        void buscaPorNome_deveBuscarCorretamente() throws SQLException {
        Sistema sistema = new Sistema(1, "Tormenta");
        Raca raca = new Raca(15, "Teste", "D", 0,0,0,0,0,0,sistema);
        RacaDBDAO racaDB = new RacaDBDAO();
        racaDB.remover(raca);
        racaDB.insere(raca);
        Raca racaBusca = racaDB.buscaPorNome("Teste");
        String nome = racaBusca.getNomeRaca();
        assertEquals("Teste", nome, "O nome da raca registrada deve ser Teste");
        }
}
```

MenusControllerTest.java

```
package com.trabalhojava.sistemarpg.controller;

import com.trabalhojava.sistemarpg.dao.ClasseDBDAO;
import com.trabalhojava.sistemarpg.dao.SistemaDBDAO;
import com.trabalhojava.sistemarpg.model.Classe;
import com.trabalhojava.sistemarpg.model.Personagem;
import com.trabalhojava.sistemarpg.model.Sistema;
import org.junit.jupiter.api.Test;

import java.sql.SQLException;
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

public class MenusControllerTest {

        @Test
        public void listaPersonagens_DeveSerInicializadaVazia() {

        MenusController controller = new MenusController();
        assertNotNull(controller.personagens, "A lista de personagens deve ser
inicializada.");
        assertTrue(controller.personagens.isEmpty(), "A lista de personagens deve estar
vazia ao iniciar.");
        }
```

```
	@Test
	public void adicionarPersonagem_DeveInserirNaLista() {

	MenusController controller = new MenusController();
	Personagem personagem = new Personagem(0,"Mago", "Mestre dos feitiços",
"teste",0,0,0,0,0,0,0);

	controller.personagens.add(personagem);

	assertEquals(1, controller.personagens.size(), "A lista deve conter exatamente 1
personagem após adicionar.");
	assertEquals("Mago", controller.personagens.getFirst().getNome(), "O personagem
adicionado deve ser 'Mago'.");
	}


	@Test
	public void gradienteRGB_DeveCalcularOffsetCorretamente() {
	double currentTime = System.currentTimeMillis() % 3000;
	double offset = currentTime / 3000.0;

	assertTrue(offset >= 0.0 && offset <= 1.0, "Offset deve estar entre 0.0 e 1.0.");
	}

	@Test
	public void removerPersonagem_DeveReduzirTamanhoDaLista() {
	MenusController controller = new MenusController();
	Personagem personagem = new Personagem(0,"Arqueiro", "Especialista em
ataques à distância", "teste", 0, 0, 0, 0, 0, 0, 0);

	controller.personagens.add(personagem);
	assertEquals(1, controller.personagens.size(), "A lista deve conter 1 personagem
após adicionar.");

	controller.personagens.remove(personagem);
	assertTrue(controller.personagens.isEmpty(), "A lista deve estar vazia após remover
o personagem.");
	}

	@Test
	void criarEDeletarPersonagem() {
	MenusController controller = new MenusController();
	Personagem personagem = new Personagem(1, "Teste", "Bem forte", "", 1, 1, 1, 1, 1,
1, 1);
	controller.personagens.add(personagem);
	assertEquals(1, controller.personagens.size(), "A lista deve conter exatamente 1
personagem após adicionar.");
	controller.personagens.remove(personagem);
```

```
        assertTrue(controller.personagens.isEmpty(), "A lista deve estar vazia após remover
o personagem.");
        }

        @Test
        void criarEDeletarClasse() throws SQLException {
        Sistema sistema = new Sistema(1, "Tormenta");
        Classe classe = new Classe(9, "Monge", "Descrição do Monge", 1, 1 , 1, sistema);
        ClasseDBDAO classeDB = new ClasseDBDAO();
        classeDB.remover(classe);
        classeDB.insere(classe);
        Classe classeBusca = classeDB.buscaPorNome("Monge");
        assertEquals(9, classeBusca.getClasseId(), "O nome da classe registrada deve ser
Monge");
        classeDB.remover(classe);
        }


        @Test
        void criarEAlterarNomeSistema() throws SQLException {
        Sistema sistema = new Sistema(3, "Sistema Antigo");
        SistemaDBDAO sistemaDB = new SistemaDBDAO();
        sistemaDB.remover(sistema);
        sistemaDB.insere(sistema);
        Sistema sistemaBusca = sistemaDB.buscaPorCodigo(3);
        assertEquals("Sistema Antigo", sistemaBusca.getNome(), "O nome do sistema
registrado deve ser 'Sistema Antigo'");
        sistemaBusca.setNome("Sistema Novo");
        sistemaDB.atualizar(sistemaBusca);
        Sistema sistemaAtualizado = sistemaDB.buscaPorCodigo(3);
        assertEquals("Sistema Novo", sistemaAtualizado.getNome(), "O nome do sistema
atualizado deve ser 'Sistema Novo'");
        sistemaDB.remover(sistemaAtualizado);
        }

        @Test
        void criarEListarSistema() throws SQLException {
        Sistema sistema = new Sistema(2, "Novo Sistema");
        SistemaDBDAO sistemaDB = new SistemaDBDAO();
        sistemaDB.remover(sistema);
        sistemaDB.insere(sistema);
        List<Sistema> sistemas = sistemaDB.listar();
        boolean sistemaEncontrado = sistemas.stream().anyMatch(s ->
s.getNome().equals("Novo Sistema"));
        assertTrue(sistemaEncontrado, "O sistema 'Novo Sistema' deve estar presente na
listagem.");
        sistemaDB.remover(sistema);
        }
```

```
}
```