InhousePartTest.class:

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package com.example.demo.domain;

import java.io.PrintStream;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import java.util.Set;
import javax.validation.ConstraintViolation;
import javax.validation.Validation;
import javax.validation.Validator;
import javax.validation.ValidatorFactory;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class InhousePartTest {
   InhousePart ip;
   private Validator validator;

   InhousePartTest() {
   }

   @BeforeEach
   void setUp() {
       this.ip = new InhousePart();
       ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
       this.validator = factory.getValidator();
   }

   @Test
   void getPartId() {
       int idValue = 4;
       this.ip.setPartId(idValue);
       Assertions.assertEquals(this.ip.getPartId(), idValue);
   }

   @Test
   void setPartId() {
       int idValue = 4;
       this.ip.setPartId(idValue);
       Assertions.assertEquals(this.ip.getPartId(), idValue);
   }

   @Test
   void testConstructor() {
       LocalDateTime now = LocalDateTime.now().truncatedTo(ChronoUnit.SECONDS);
       InhousePart ip = new InhousePart("Sample Part", 100.0, 10, 5, 20, now, 18865);
       Assertions.assertEquals("Sample Part", ip.getName());
       Assertions.assertEquals(100.0, ip.getPrice());
       Assertions.assertEquals(10, ip.getInv());
       Assertions.assertEquals(5, ip.getMin());
       Assertions.assertEquals(20, ip.getMax());
       Assertions.assertEquals(now, ip.getDateAdded().truncatedTo(ChronoUnit.SECONDS));
```

```
        Assertions.assertEquals(18865, ip.getStoreNumber());
    }

    @Test
    void testInvalidInventory() {
        this.ip.setInv(-10);
        Set<ConstraintViolation<InhousePart>> violations = this.validator.validate(this.ip, new
Class[0]);
        violations.forEach((violation) -> {
            PrintStream var10000 = System.out;
            String var10001 = String.valueOf(violation.getPropertyPath());
            var10000.println("Violation: " + var10001 + " - " + violation.getMessage());
        });
        Assertions.assertTrue(violations.stream().anyMatch((v) -> {
            return v.getPropertyPath().toString().equals("inv") &&
v.getMessage().contains("Inventory value must be positive");
        }), "Inventory should not be negative.");
    }

    @Test
    void testDateAddedAndStoreNumber() {
        LocalDateTime now = LocalDateTime.now();
        this.ip.setDateAdded(now);
        this.ip.setStoreNumber(1001);
        Assertions.assertEquals(now, this.ip.getDateAdded(), "DateAdded should match the
assigned value.");
        Assertions.assertEquals(1001, this.ip.getStoreNumber(), "StoreNumber should match the
assigned value.");
    }

    @Test
    void testPartId() {
        int partId = 1234;
        this.ip.setPartId(partId);
        Assertions.assertEquals(partId, this.ip.getPartId(), "Part ID should match the assigned
value.");
    }

    @Test
    void testSettersAndGetters() {
        this.ip.setName("New Part Name");
        this.ip.setPrice(150.0);
        this.ip.setInv(15);
        this.ip.setMin(10);
        this.ip.setMax(25);
        Assertions.assertEquals("New Part Name", this.ip.getName());
        Assertions.assertEquals(150.0, this.ip.getPrice());
        Assertions.assertEquals(15, this.ip.getInv());
        Assertions.assertEquals(10, this.ip.getMin());
        Assertions.assertEquals(25, this.ip.getMax());
    }
}
```

OutsourcedPartTest.class:

```
//
// Source code recreated from a .class file by IntelliJ IDEA
```

```java
// (powered by FernFlower decompiler)
//

package com.example.demo.domain;

import java.util.Set;
import javax.validation.ConstraintViolation;
import javax.validation.Validation;
import javax.validation.Validator;
import javax.validation.ValidatorFactory;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class OutsourcedPartTest {
   OutsourcedPart op;
   private Validator validator;

   OutsourcedPartTest() {
   }

   @BeforeEach
   void setUp() {
       this.op = new OutsourcedPart();
       ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
       this.validator = factory.getValidator();
   }

   @Test
   void getCompanyName() {
       String name = "test company name";
       this.op.setCompanyName(name);
       Assertions.assertEquals(name, this.op.getCompanyName());
   }

   @Test
   void setCompanyName() {
       String name = "test company name";
       this.op.setCompanyName(name);
       Assertions.assertEquals(name, this.op.getCompanyName());
   }

   @Test
   void testNullCompanyName() {
       this.op.setCompanyName((String)null);
       Set<ConstraintViolation<OutsourcedPart>> violations =
this.validator.validate(this.op, new Class[0]);
       Assertions.assertTrue(violations.stream().anyMatch((v) -> {
```

```
            return v.getPropertyPath().toString().equals("companyName") &&
v.getMessage().contains("must not be null");
        }), "Company name should not be null.");
    }

    @Test
    void testInvalidStoreNumber() {
        this.op.setStoreNumber(-1);
        Set<ConstraintViolation<OutsourcedPart>> violations =
this.validator.validate(this.op, new Class[0]);
        Assertions.assertTrue(violations.stream().anyMatch((v) -> {
            return v.getPropertyPath().toString().equals("storeNumber") &&
v.getMessage().contains("must be greater than or equal to 0");
        }), "Store number should not be negative.");
    }
}
```

PartTest.class:

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package com.example.demo.domain;

import java.io.PrintStream;
import java.time.LocalDateTime;
import java.util.HashSet;
import java.util.Set;
import javax.validation.ConstraintViolation;
import javax.validation.Validation;
import javax.validation.Validator;
import javax.validation.ValidatorFactory;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class PartTest {
    private Validator validator;
    private Part partIn;
    private Part partOut;

    PartTest() {
    }

    @BeforeEach
    void setUp() {
        this.partIn = new InhousePart();
```

```
        this.partOut = new OutsourcedPart();
        ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
        this.validator = factory.getValidator();
    }

    @Test
    void getId() {
        Long idValue = 4L;
        this.partIn.setId(idValue);
        Assertions.assertEquals(idValue, this.partIn.getId());
        this.partOut.setId(idValue);
        Assertions.assertEquals(idValue, this.partOut.getId());
    }

    @Test
    void setId() {
        Long idValue = 4L;
        this.partIn.setId(idValue);
        Assertions.assertEquals(idValue, this.partIn.getId());
        this.partOut.setId(idValue);
        Assertions.assertEquals(idValue, this.partOut.getId());
    }

    @Test
    void getName() {
        String name = "test inhouse part";
        this.partIn.setName(name);
        Assertions.assertEquals(name, this.partIn.getName());
        name = "test outsourced part";
        this.partOut.setName(name);
        Assertions.assertEquals(name, this.partOut.getName());
    }

    @Test
    void setName() {
        String name = "test inhouse part";
        this.partIn.setName(name);
        Assertions.assertEquals(name, this.partIn.getName());
        name = "test outsourced part";
        this.partOut.setName(name);
        Assertions.assertEquals(name, this.partOut.getName());
    }

    @Test
    void getPrice() {
        double price = 1.0;
        this.partIn.setPrice(price);
        Assertions.assertEquals(price, this.partIn.getPrice());
        this.partOut.setPrice(price);
```

```
        Assertions.assertEquals(price, this.partOut.getPrice());
    }

    @Test
    void setPrice() {
        double price = 1.0;
        this.partIn.setPrice(price);
        Assertions.assertEquals(price, this.partIn.getPrice());
        this.partOut.setPrice(price);
        Assertions.assertEquals(price, this.partOut.getPrice());
    }

    @Test
    void getInv() {
        int inv = 5;
        this.partIn.setInv(inv);
        Assertions.assertEquals(inv, this.partIn.getInv());
        this.partOut.setInv(inv);
        Assertions.assertEquals(inv, this.partOut.getInv());
    }

    @Test
    void setInv() {
        int inv = 5;
        this.partIn.setInv(inv);
        Assertions.assertEquals(inv, this.partIn.getInv());
        this.partOut.setInv(inv);
        Assertions.assertEquals(inv, this.partOut.getInv());
    }

    @Test
    void getStoreNumber() {
        int storeNumber = 101;
        this.partIn.setStoreNumber(storeNumber);
        Assertions.assertEquals(storeNumber, this.partIn.getStoreNumber());
        this.partOut.setStoreNumber(storeNumber);
        Assertions.assertEquals(storeNumber, this.partOut.getStoreNumber());
    }

    @Test
    void setStoreNumber() {
        int storeNumber = 101;
        this.partIn.setStoreNumber(storeNumber);
        Assertions.assertEquals(storeNumber, this.partIn.getStoreNumber());
        this.partOut.setStoreNumber(storeNumber);
        Assertions.assertEquals(storeNumber, this.partOut.getStoreNumber());
    }

    @Test
```

```
    void getDateAdded() {
        LocalDateTime dateAdded = LocalDateTime.now();
        this.partIn.setDateAdded(dateAdded);
        Assertions.assertEquals(dateAdded, this.partIn.getDateAdded());
        this.partOut.setDateAdded(dateAdded);
        Assertions.assertEquals(dateAdded, this.partOut.getDateAdded());
    }

    @Test
    void setDateAdded() {
        LocalDateTime dateAdded = LocalDateTime.now();
        this.partIn.setDateAdded(dateAdded);
        Assertions.assertEquals(dateAdded, this.partIn.getDateAdded());
        this.partOut.setDateAdded(dateAdded);
        Assertions.assertEquals(dateAdded, this.partOut.getDateAdded());
    }

    @Test
    void getProducts() {
        Product product1 = new Product();
        Product product2 = new Product();
        Set<Product> myProducts = new HashSet();
        myProducts.add(product1);
        myProducts.add(product2);
        this.partIn.setProducts(myProducts);
        Assertions.assertEquals(myProducts, this.partIn.getProducts());
        this.partOut.setProducts(myProducts);
        Assertions.assertEquals(myProducts, this.partOut.getProducts());
    }

    @Test
    void setProducts() {
        Product product1 = new Product();
        Product product2 = new Product();
        Set<Product> myProducts = new HashSet();
        myProducts.add(product1);
        myProducts.add(product2);
        this.partIn.setProducts(myProducts);
        Assertions.assertEquals(myProducts, this.partIn.getProducts());
        this.partOut.setProducts(myProducts);
        Assertions.assertEquals(myProducts, this.partOut.getProducts());
    }

    @Test
    void testToString() {
        String name = "test inhouse part";
        this.partIn.setName(name);
        Assertions.assertEquals(name, this.partIn.toString());
        name = "test outsourced part";
```

```
        this.partOut.setName(name);
        Assertions.assertEquals(name, this.partOut.toString());
    }

    @Test
    void testEquals() {
        this.partIn.setId(1L);
        Part newPartIn = new InhousePart();
        ((Part)newPartIn).setId(1L);
        Assertions.assertEquals(this.partIn, newPartIn);
        this.partOut.setId(1L);
        Part newPartOut = new OutsourcedPart();
        ((Part)newPartOut).setId(1L);
        Assertions.assertEquals(this.partOut, newPartOut);
    }

    @Test
    void testNullName() {
        this.partIn.setName((String)null);
        Set<ConstraintViolation<Part>> violations =
this.validator.validate(this.partIn, new Class[0]);
        violations.forEach((violation) -> {
            PrintStream var10000 = System.out;
            String var10001 = String.valueOf(violation.getPropertyPath());
            var10000.println("Violation: " + var10001 + " - " +
violation.getMessage());
        });
        Assertions.assertTrue(violations.stream().anyMatch((v) -> {
            return v.getPropertyPath().toString().equals("name") &&
v.getMessage().contains("Name cannot be null");
        }), "Name should not be null.");
    }

    @Test
    void testInvalidPrice() {
        this.partIn.setPrice(-5.0);
        Set<ConstraintViolation<Part>> violations =
this.validator.validate(this.partIn, new Class[0]);
        violations.forEach((violation) -> {
            PrintStream var10000 = System.out;
            String var10001 = String.valueOf(violation.getPropertyPath());
            var10000.println("Violation: " + var10001 + " - " +
violation.getMessage());
        });
        Assertions.assertTrue(violations.stream().anyMatch((v) -> {
            return v.getPropertyPath().toString().equals("price") &&
v.getMessage().contains("Price value must be positive");
        }), "Price should not be negative.");
    }
```

```
    @Test
    void testEmptyProductList() {
        this.partIn.setProducts(new HashSet());
        Assertions.assertTrue(this.partIn.getProducts().isEmpty(), "Product list
should be empty.");
    }

    @Test
    void testHashCode() {
        this.partIn.setId(1L);
        this.partOut.setId(1L);
        Assertions.assertEquals(this.partIn.hashCode(),
this.partOut.hashCode());
    }

    @Test
    void testMinInventory() {
        Part part = new InhousePart("Sample Part", 10.0, 4, 5, 10,
LocalDateTime.now(), 1);
        Set<ConstraintViolation<Part>> violations =
this.validator.validate(part, new Class[0]);
        Assertions.assertTrue(violations.stream().anyMatch((v) -> {
            return v.getPropertyPath().toString().equals("inv") &&
v.getMessage().equals("Cannot enter inventory outside of Min/Max range");
        }), "Inventory should not go below the minimum value.");
    }

    @Test
    void testMaxInventory() {
        Part part = new InhousePart("Sample Part", 10.0, 15, 5, 10,
LocalDateTime.now(), 2);
        Set<ConstraintViolation<Part>> violations =
this.validator.validate(part, new Class[0]);
        Assertions.assertTrue(violations.stream().anyMatch((v) -> {
            return v.getPropertyPath().toString().equals("inv") &&
v.getMessage().equals("Cannot enter inventory outside of Min/Max range");
        }), "Inventory should not exceed the maximum value.");
    }
}
```

ProductTest.class:

```
package com.example.demo.domain;

import com.example.demo.service.ProductService;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
```

```java
import org.mockito.MockitoAnnotations;

import javax.validation.ConstraintViolation;
import javax.validation.Validation;
import javax.validation.Validator;
import javax.validation.ValidatorFactory;
import java.util.HashSet;
import java.util.Set;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.doNothing;


class ProductTest {
   Product product;
   private Validator validator;

   @Mock
   private ProductService productService;  // Mocking ProductService

   @BeforeEach
   public void setUp(){
       MockitoAnnotations.initMocks(this);  // Initialize mocks

       product = new Product();
       ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
       validator = factory.getValidator();
   }


   @Test
   void getId() {
       Long idValue=4L;
       product.setId(idValue);
       assertEquals(product.getId(), idValue);
   }

   @Test
   void setId() {
       Long idValue=4L;
       product.setId(idValue);
       assertEquals(product.getId(), idValue);
   }

   @Test
   void getName() {
       String name="test product";
       product.setName(name);
       assertEquals(name,product.getName());
```

```
}

@Test
void setName() {
    String name="test product";
    product.setName(name);
    assertEquals(name,product.getName());
}

@Test
void getPrice() {
    double price=1.0;
    product.setPrice(price);
    assertEquals(price,product.getPrice());
}

@Test
void setPrice() {
    double price=1.0;
    product.setPrice(price);
    assertEquals(price,product.getPrice());
}

@Test
void getInv() {
    int inv=5;
    product.setInv(inv);
    assertEquals(inv,product.getInv());
}

@Test
void setInv() {
    int inv=5;
    product.setInv(inv);
    assertEquals(inv,product.getInv());
}

@Test
void getParts() {
    Part part1 = new OutsourcedPart();
    Part part2 = new InhousePart();
    Set<Part> myParts= new HashSet<>();
    myParts.add(part1);
    myParts.add(part2);
    product.setParts(myParts);
    assertEquals(myParts,product.getParts());
}

@Test
```

```
    void setParts() {
        Part part1 = new OutsourcedPart();
        Part part2 = new InhousePart();
        Set<Part> myParts= new HashSet<>();
        myParts.add(part1);
        myParts.add(part2);
        product.setParts(myParts);
        assertEquals(myParts,product.getParts());
    }

    @Test
    void testToString() {
        String name="test product";
        product.setName(name);
        assertEquals(name,product.toString());
    }

    @Test
    void testEquals() {
        product.setId(1l);
        Product newProduct= new Product();
        newProduct.setId(1l);
        assertEquals(product,newProduct);
    }

    @Test
    void testHashCode() {
        product.setId(1l);
        Product newProduct= new Product();
        newProduct.setId(1l);
        assertEquals(product.hashCode(),newProduct.hashCode());
    }
    @Test
    void testNullName() {
        product.setName(null);  // Set name to null
        Set<ConstraintViolation<Product>> violations =
validator.validate(product);

        violations.forEach(v -> {
            System.out.println("Violation: " + v.getPropertyPath() + " - " +
v.getMessage());
        });

        Assertions.assertTrue(violations.stream().anyMatch(v ->
                v.getPropertyPath().toString().equals("name") &&
                        v.getMessage().contains("Product name cannot be null")
        ), "Product name should not be null.");
    }
```

```
    @Test
    void testInventoryOutsideRange() {
        product.setInv(1000); // Exceeds maximum allowed inventory
        Set<ConstraintViolation<Product>> violations =
validator.validate(product);

        violations.forEach(v -> {
            System.out.println("Violation: " + v.getPropertyPath() + " - " +
v.getMessage());
        });

        Assertions.assertTrue(violations.stream().anyMatch(v ->
                v.getPropertyPath().toString().equals("inv") &&
                        v.getMessage().contains("Inventory cannot exceed 999")
// Ensure this matches the actual message
        ), "Inventory should not exceed the allowed maximum.");
    }


    @Test
    void testDateAddedAutomaticallySet() {
        // Create a new Product instance (simulate creating a new product)
        Product product = new Product();
        product.setName("Test Product");
        product.setPrice(100.0);
        product.setInv(10);
        product.setStoreNumber(123);

        // Validate the product
        Set<ConstraintViolation<Product>> violations =
this.validator.validate(product);

        // Check if there are any violations (there should be none related to
dateAdded being null)
        Assertions.assertTrue(violations.isEmpty(), "There should be no
violations.");

        // Manually trigger the @PrePersist lifecycle event
        product.onCreate();

        // Simulate saving the product (no need to mock here, we're just testing
the date)
        Assertions.assertNotNull(product.getDateAdded(), "DateAdded should be
automatically set before saving.");
    }
```

```
    @Test
    void testAddPart() {
        Part part = new InhousePart();
        product.getParts().add(part);
        Assertions.assertTrue(product.getParts().contains(part), "Product should
contain the added part.");
    }



}
```