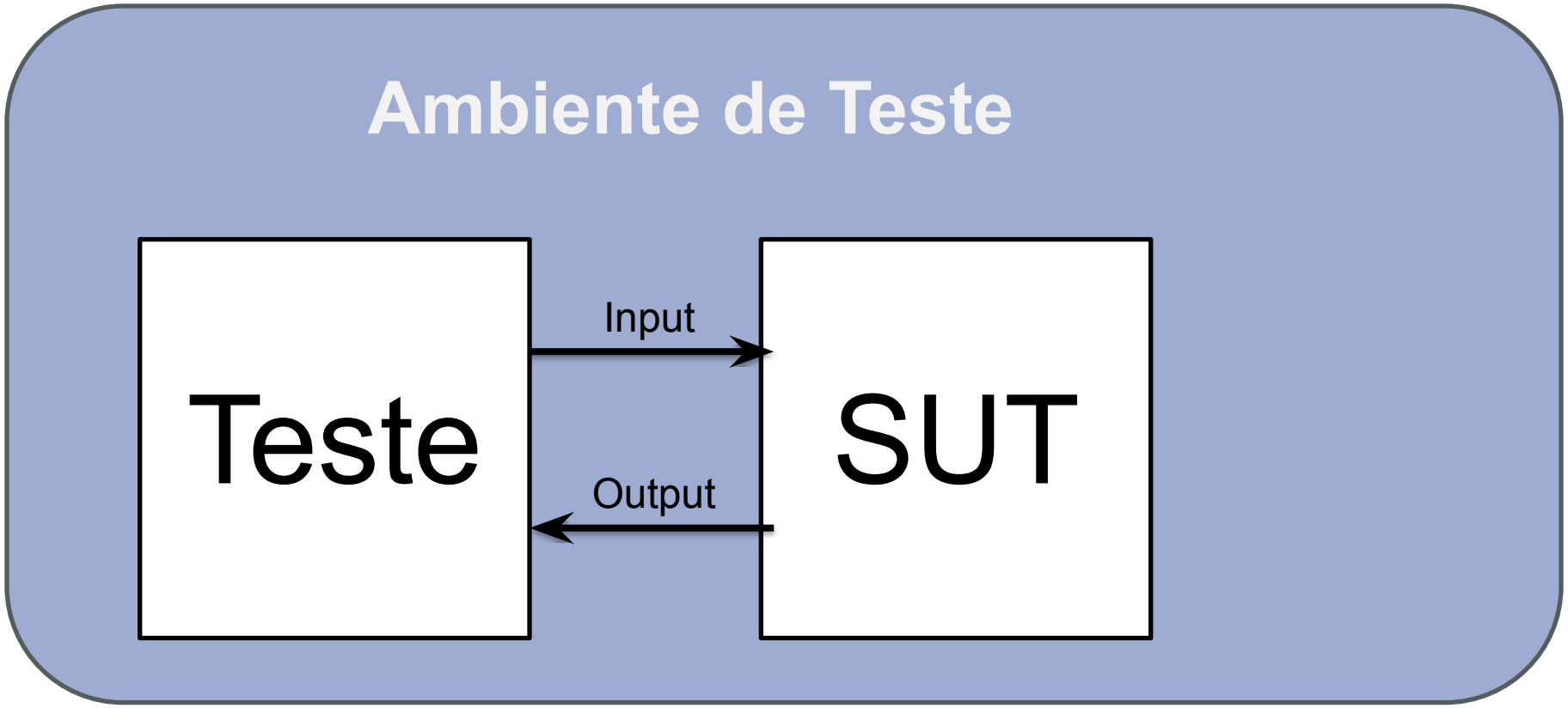
Residência TRF5
Gerência de Configuração e Teste de Software

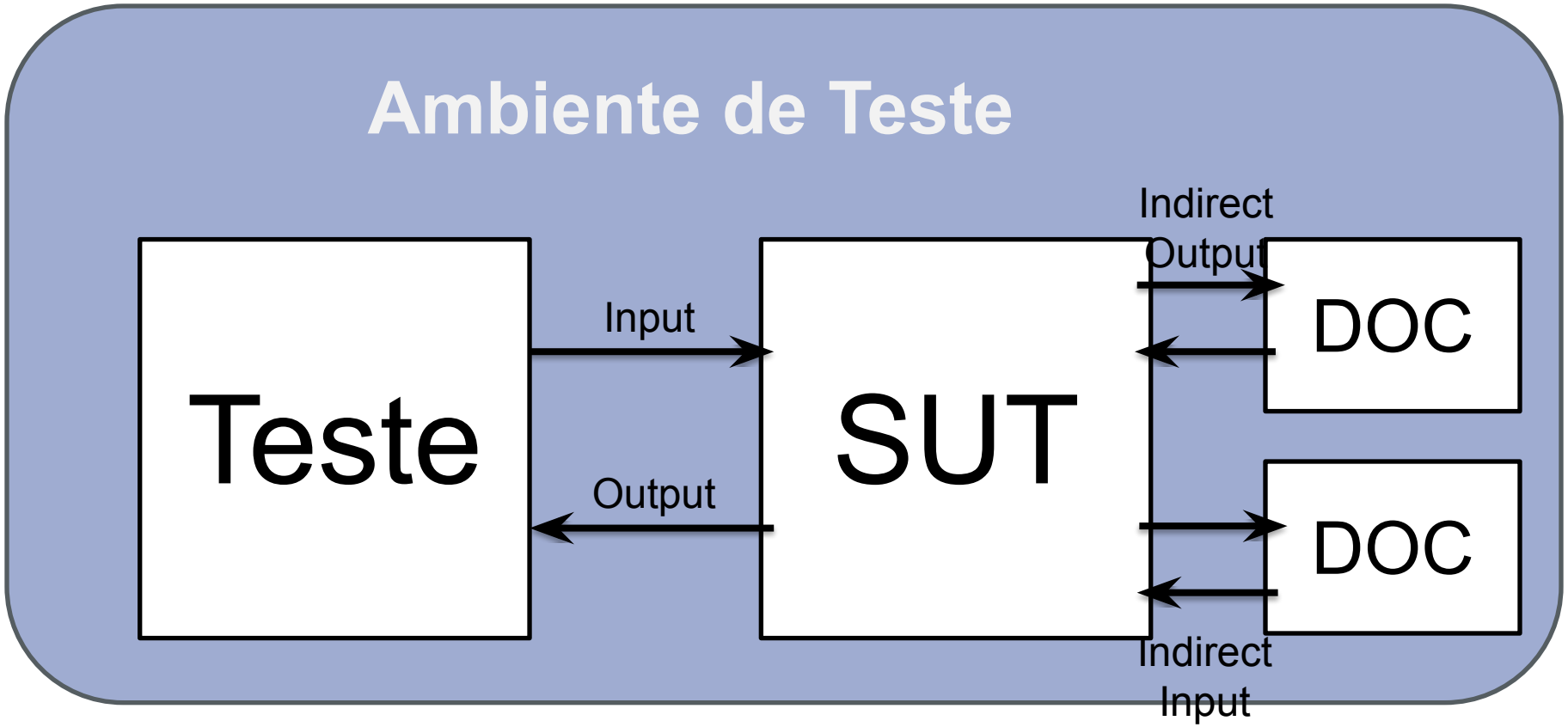
Prof. Eiji Adachi

Objetivos

- Apresentar conceitos relacionados a Test Doubles

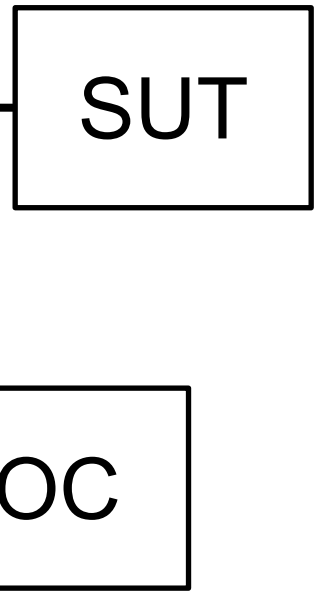


SUT = System Under Test



DOC = Depended-on Component

```java
public static String readContent(String dir, String fileName) throws IOException {
	final Path path = Paths.get(dir, fileName);

	try ( BufferedReader reader = Files.newBufferedReader(path) ) {
		final StringBuilder builder = new StringBuilder();
		String line;

		while ((line = reader.readLine()) != null) {
			builder.append(line);
		}

		return builder.toString();
	} catch (IOException e) {
		LOGGER.log(Level.INFO, "Error while processing " + dir + fileName, e);
		throw e;
	}
}
```



```
@Test
public void testCorrectReading() throws IOException {
        String actualContent = readContent("/test/resources/", "355685.csv");

        assertEquals(expectedContet, actualContent);
}
```

Desafios

- DOCs podem retornar valores que afetam o comportamento do SUT, mas nem sempre é fácil, a partir da classe de testes, forçar que os DOCs retornem alguns valores que afetam o comportamento do SUT
  - Ex.: DOCs são recursos externos que podem lançar uma exceção e isto afetar o comportamento do SUT, mas como forçar, a partir dos testes, uma exceção num recurso externo?

```
@Test
public void testCorrectReading() throws IOException {
        String actualContent = readContent("/test/resources/", "355685.csv");

        assertEquals(expectedContet, actualContent);
}
```

```
@Test
public void testReadingNonExistentFile() throws IOException {
        assertThrows(NoSuchFileException.class, () -> {readContent("/test/resources/", "nonexistent.file”});
}
```

```
@Test
public void testCorrectReading() throws IOException {
        String actualContent = readContent("/test/resources/", "355685.csv");

        assertEquals(expectedContet, actualContent);
}
```

```
@Test
public void testReadingNonExistentFile() throws IOException {
        assertThrows(NoSuchFileException.class, () -> {readContent("/test/resources/",
"nonexistent.file”});
}
```

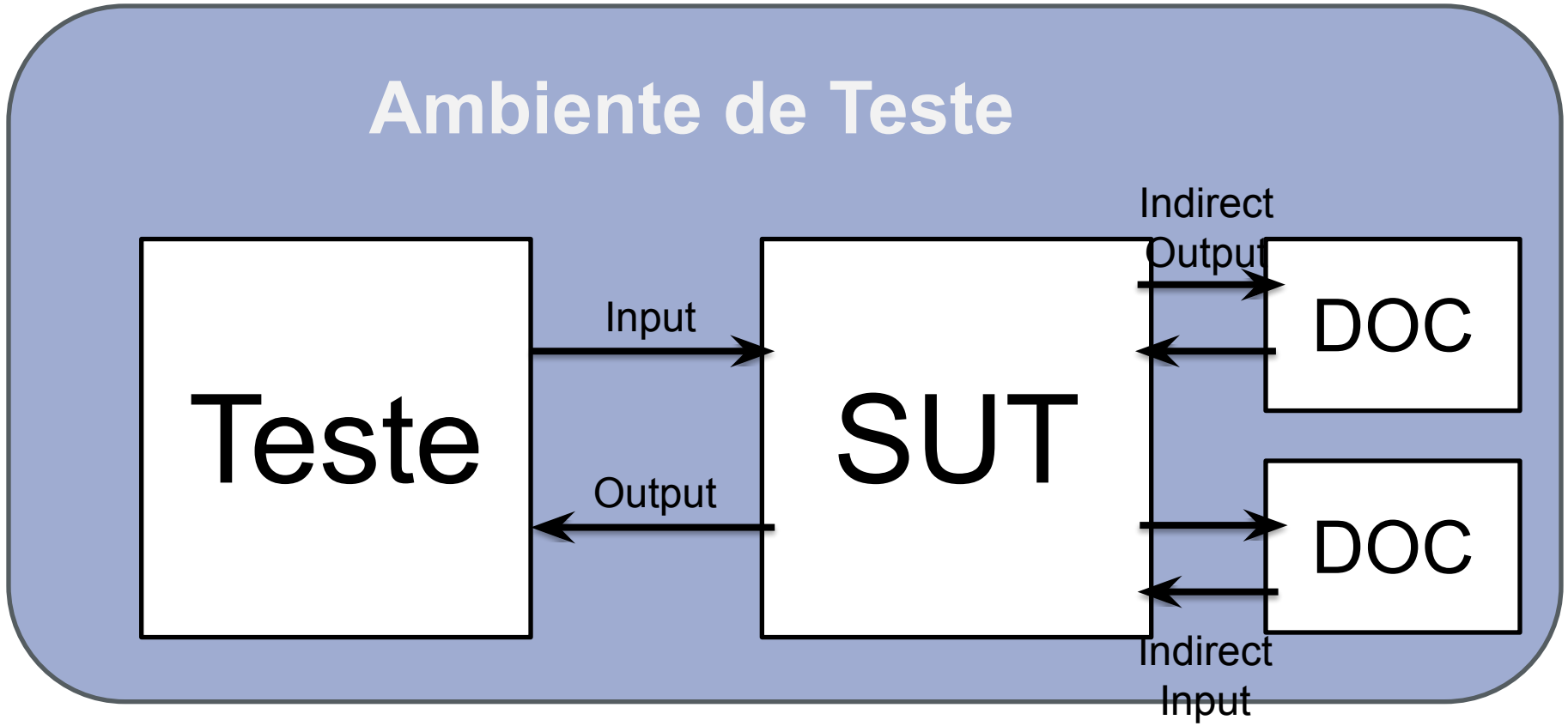
Como forçar um erro no meio da leitura do arquivo?

```
@Test
public void testErrorWhileReading() throws IOException {
        String partialContent = readContent("/test/resources/", "355685.csv");

        assertEquals(expectedPartialContent, partialContent);
}
```

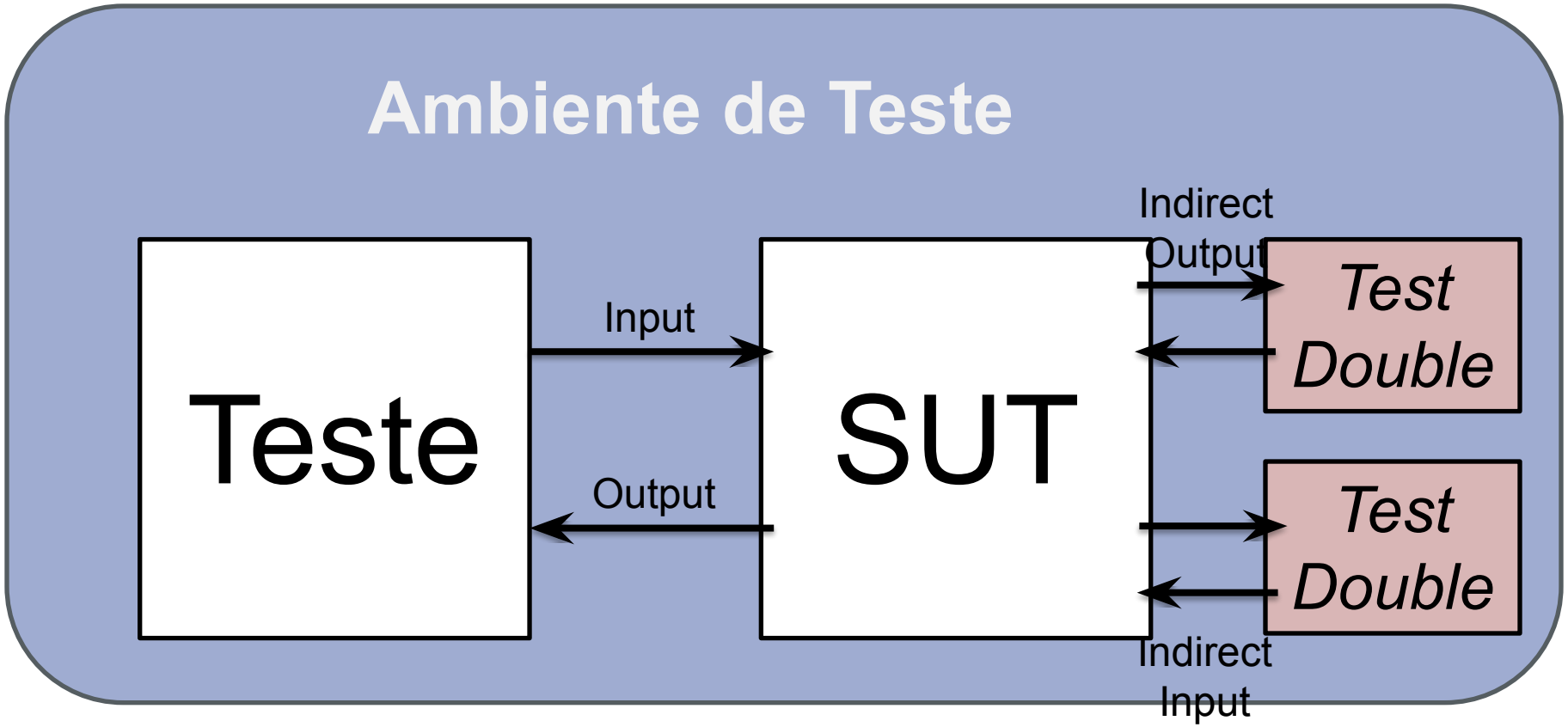
Outos desafios

- DOCs podem ser lentos, tornando custosa a execução de uma suíte de testes
  - Ex.: DOC pode ser uma classe de acesso a uma API, ou a um Banco de Dados, ou um Moto de Cálculos cujo processamento demora

Como testar?


Ambiente de Teste
Teste
Input
Output
SUT
Indirect Output
DOC
DOC
Indirect Input

Como testar?


Ambiente de Teste
Teste
Input
Output
SUT
Indirect Output
Test Double
Test Double
Indirect Input

O que é um ***Test Double*** *?*

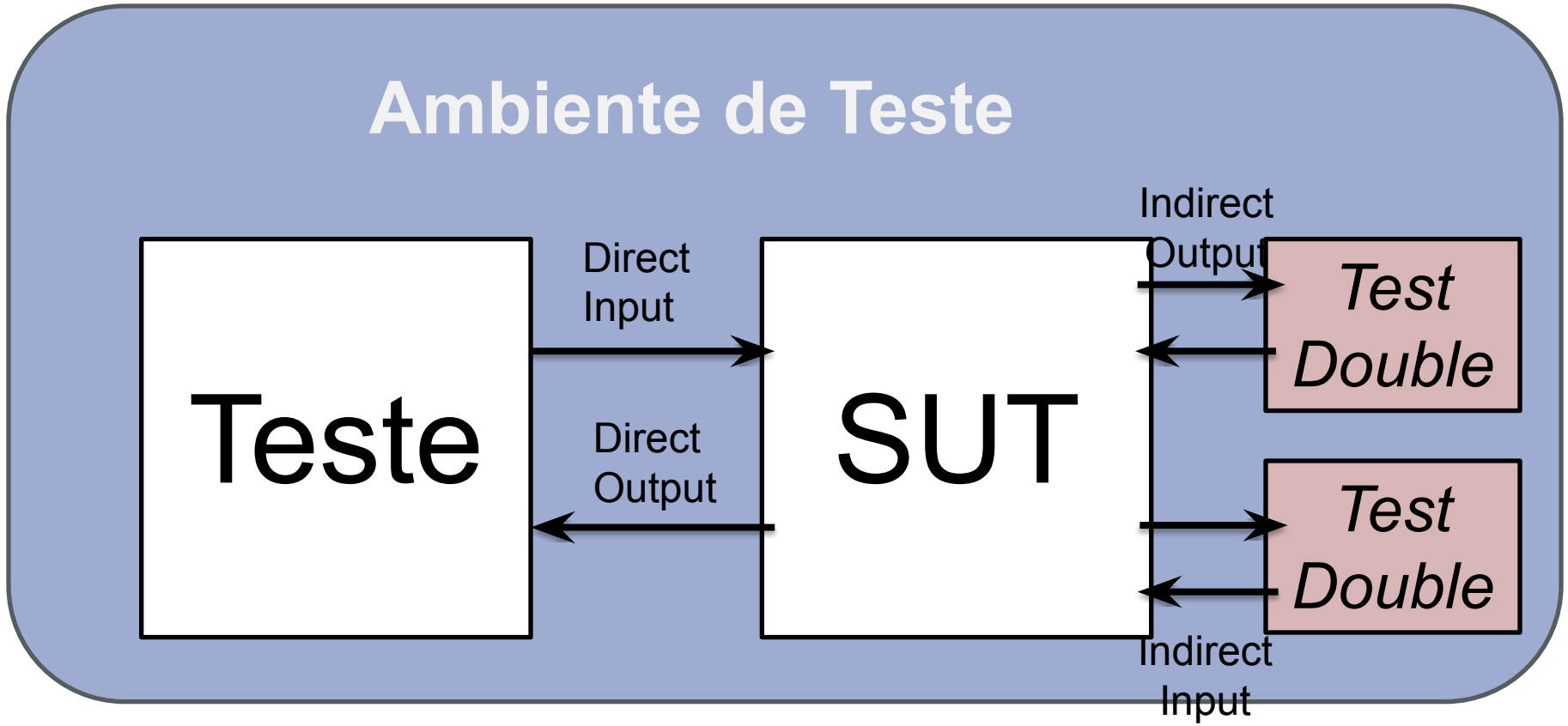
Test Double

é qualquer objeto que é instalado no lugar de outro com o propósito de ***executar um teste****.*

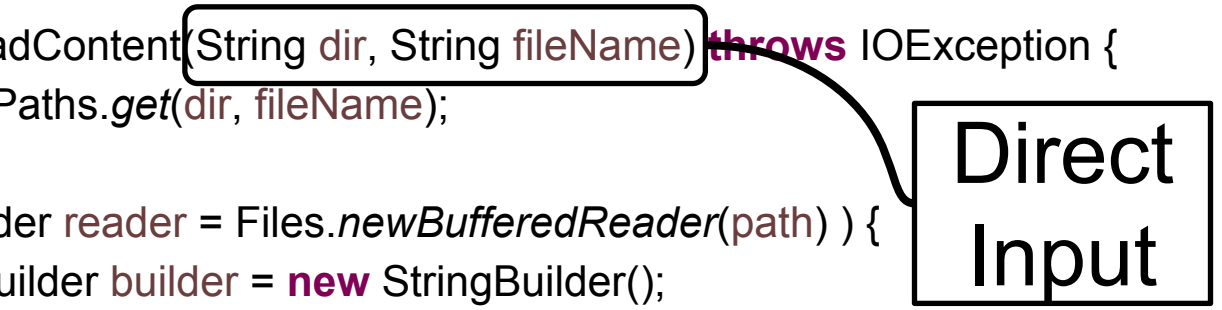
MESZAROS, Gerard. **xUnit test patterns: Refactoring test code**. Pearson Education, 2007.

Inputs e Outputs

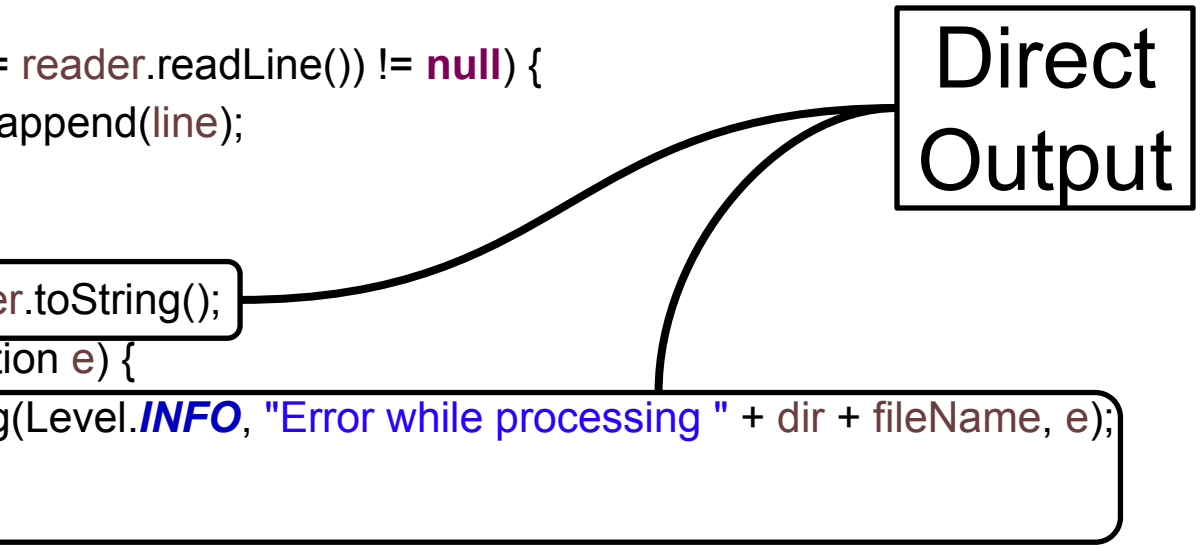
- No contexto de Test Doubles, é importante distinguir:
  - Direct Input
  - Direct Output
  - Indirect Input
  - Indirect Output


Ambiente de Teste
Teste
Direct Input
Direct Output
SUT
Indirect Output
Test Double
Test Double
Indirect Input

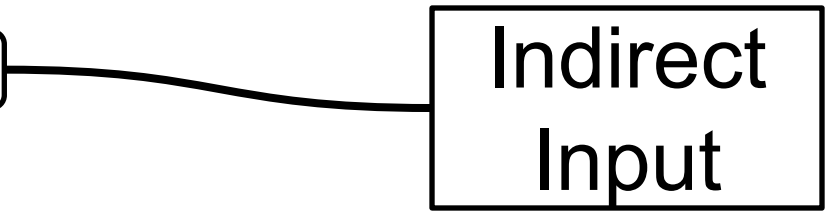
```java
public static String readContent(String dir, String fileName) throws IOException {
    final Path path = Paths.get(dir, fileName);

    try ( BufferedReader reader = Files.newBufferedReader(path) ) {
        final StringBuilder builder = new StringBuilder();
        String line;

        while ((line = reader.readLine()) != null) {
            builder.append(line);
        }

        return builder.toString();
    } catch (IOException e) {
        LOGGER.log(Level.INFO, "Error while processing " + dir + fileName, e);
        throw e;
    }
}
```



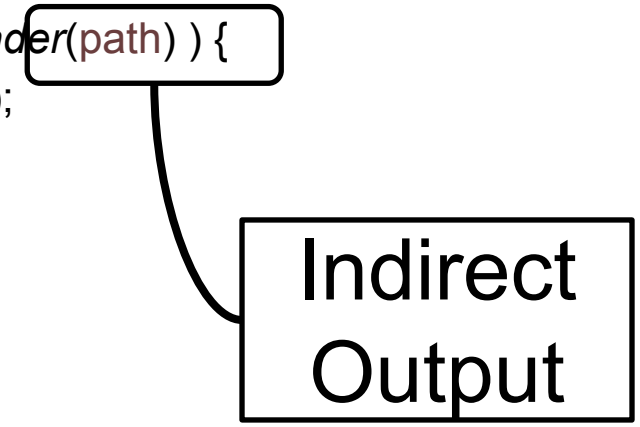
```java
public static String readContent(String dir, String fileName) throws IOException {
    final Path path = Paths.get(dir, fileName);

    try ( BufferedReader reader = Files.newBufferedReader(path) ) {
        final StringBuilder builder = new StringBuilder();
        String line;

        while ((line = reader.readLine()) != null) {
            builder.append(line);
        }

        return builder.toString();
    } catch (IOException e) {
        LOGGER.log(Level.INFO, "Error while processing " + dir + fileName, e);
        throw e;
    }
}
```



```java
public static String readContent(String dir, String fileName) throws IOException {
    final Path path = Paths.get(dir, fileName);

    try ( BufferedReader reader = Files.newBufferedReader(path) ) {
        final StringBuilder builder = new StringBuilder();
        String line;

        while ((line = reader.readLine()) != null) {
            builder.append(line);
        }

        return builder.toString();
    } catch (IOException e) {
        LOGGER.log(Level.INFO, "Error while processing " + dir + fileName, e);
        throw e;
    }
}
```



```
public static String readContent(String dir, String fileName) throws IOException {
        final Path path = Paths.get(dir, fileName);

        try ( BufferedReader reader = Files.newBufferedReader(path) ) {
                final StringBuilder builder = new StringBuilder();
                String line;

                while ((line = reader.readLine()) != null) {
                        builder.append(line);
                }

                return builder.toString();
        } catch (IOException e) {
                LOGGER.log(Level.INFO, "Error while processing " + dir + fileName, e);
                throw e;
        }
}
```



Inputs e Outputs

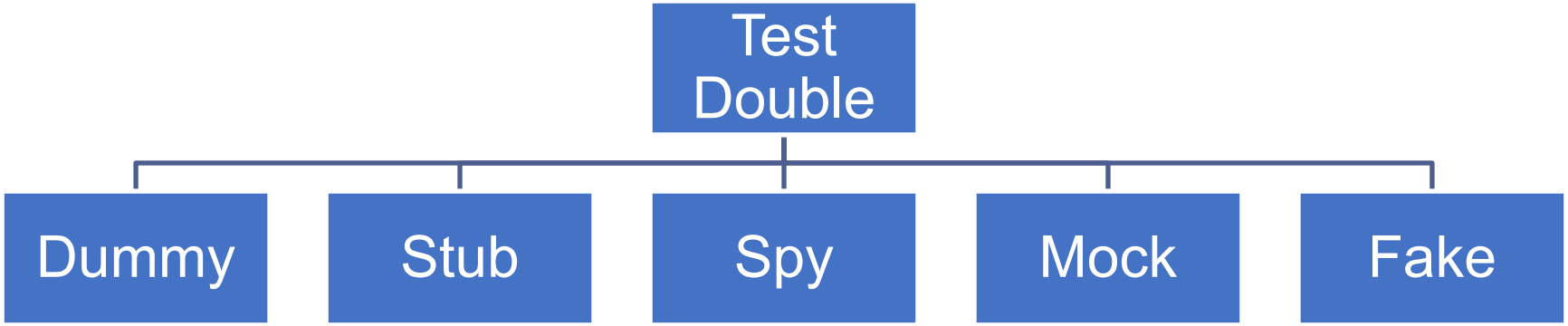
- Definição:
  - Direct Input – entrada passada diretamente para o SUT; passada do Teste para o SUT
  - Direct Output – saída retornada diretamente pelo SUT; retornada do SUT para o Teste
  - Indirect Input – entrada passada indiretamente para o SUT; passada do DOC para o SUT
  - Indirect Output – saída retornada indiretamente pelo SUT; retornada do SUT para o DOC

Quais os tipos de

Test Double

?

Tipos de Test Doubles



Dummy Object

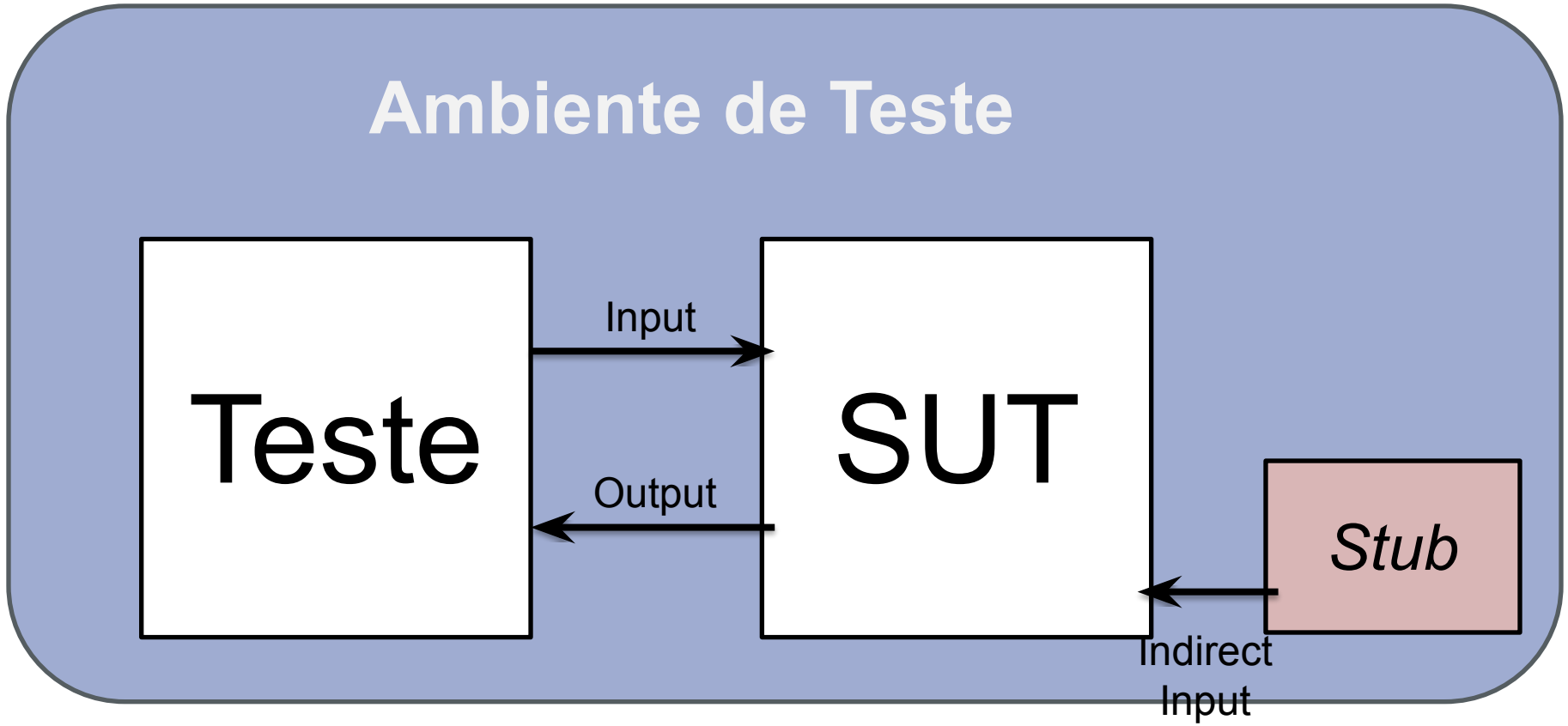
- Fazem nada, nunca são usados. Apenas precisam existir para atender alguma questão sintática da linguagem de programação.
  - Ex.: Criar um Dummy Object para atender a assinatura de um método.

```
@Test
public void testCarrinhoCheckout() {
        CarrinhoServico servico = new CarrinhoServico();
        // Inputs
        Estado estado = Estado.AC;
        double preco = 10.0d;
        double peso = 1.0d;
        Item i1 = new Item("", "", preco, peso, ItemTipo.CASA);
        List<Item> itens = Arrays.asList(i1);

        LocalDate checkoutDate = LocalDate.now();
        Endereco endereco = new Endereco("", 1, "", "", "", estado);
        Usuario usuario = new Usuario("", "", "", endereco);


        Pedido resultadoRetornado = servico.finalizar(usuario, itens, checkoutDate);


        ...
```

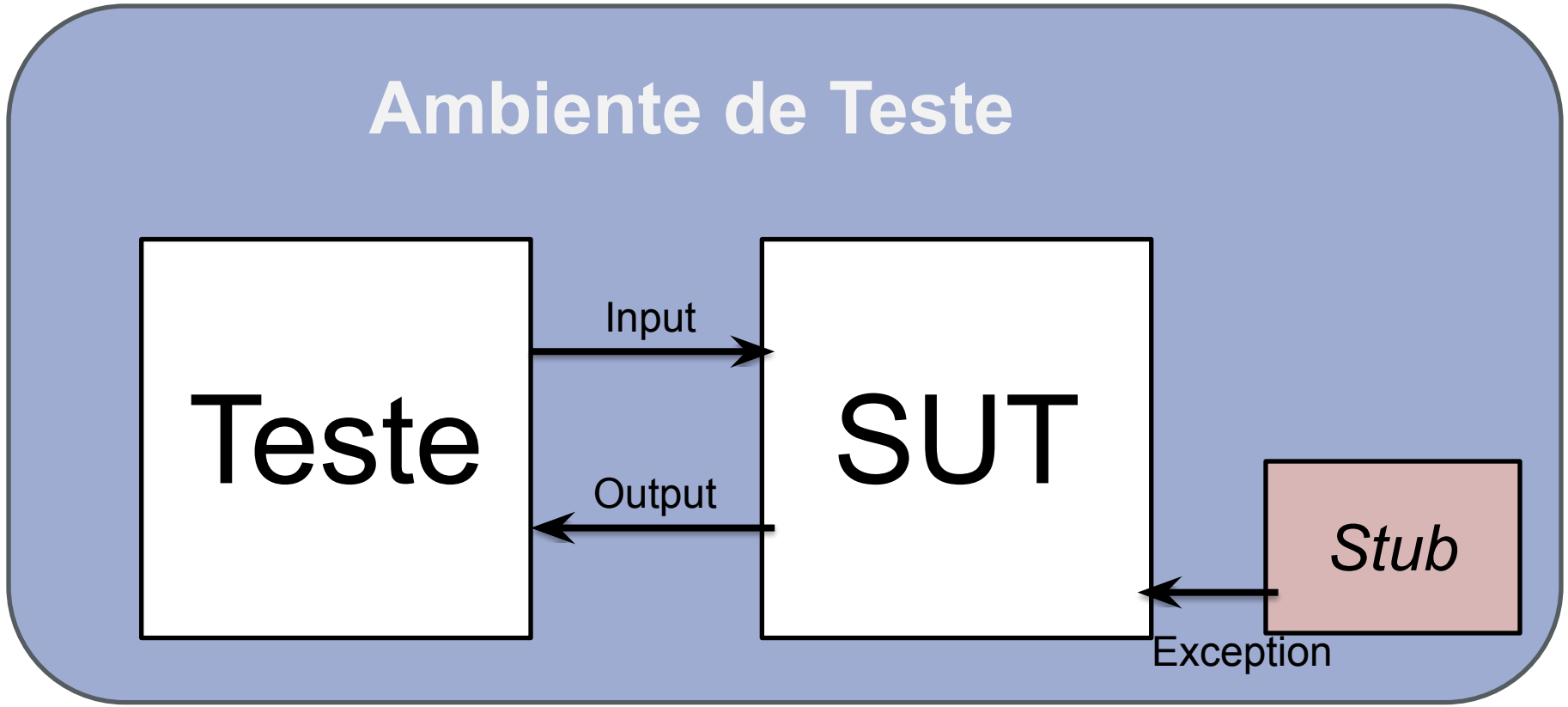
Stub

- Objeto empregado para controlar as Indirect Inputs passados para o SUT
  - Tipos:
    - Respondedor – Empregado para passar Indirect Inputs válidas e inválidas para o SUT por meio de return´s
    - Sabotador – Empregado para passar Indirect Inputs anormais para o SUT por meio de exceções

Stub

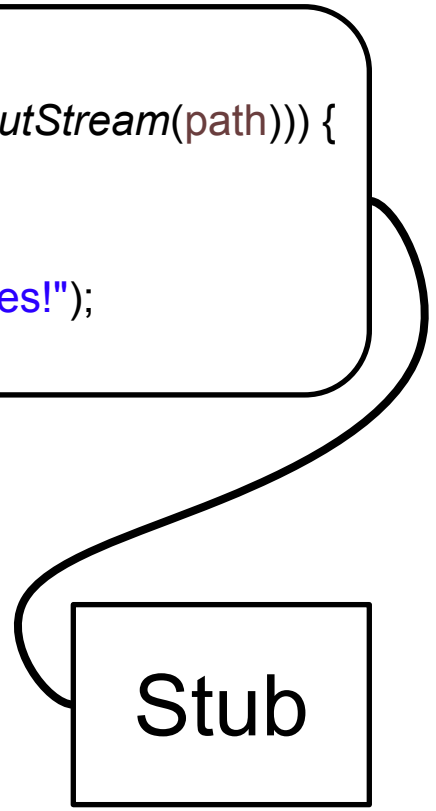

Ambiente de Teste
Teste
Input
Output
SUT
Stub
Indirect
Input

Stub

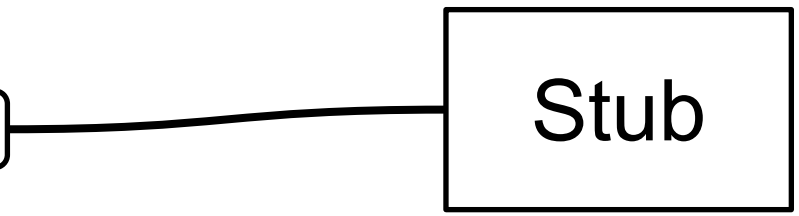


```
public static String readContent(String dir, String fileName) throws IOException {
        final Path path = Paths.get(dir, fileName);

        try ( BufferedReader reader = Files.newBufferedReader(path) ) {
                final StringBuilder builder = new StringBuilder();
                String line;

                while ((line = reader.readLine()) != null) {
                        builder.append(line);
                }

                return builder.toString();
        } catch (IOException e) {
                LOGGER.log(Level.INFO, "Error while processing " + dir + fileName, e);
                throw e;
        }
}
```

```
public static String readContent(String dir, String fileName) throws IOException {
    final Path path = Paths.get(dir, fileName);

    try (
          BufferedReader reader =
        new BufferedReader(new InputStreamReader(Files.newInputStream(path))) {
          @Override
          public String readLine() throws IOException {
                throw new IOException("Sabotage for testing purposes!");
          }
        };
        ){
        final StringBuilder builder = new StringBuilder();
        String line = null;

        while ((line = reader.readLine()) != null) {
            builder.append(line);
        }

        return builder.toString();
    } catch (IOException e) {
        LOGGER.log(Level.INFO, "Error while processing " + dir + fileName, e);
        throw e;
    }
```



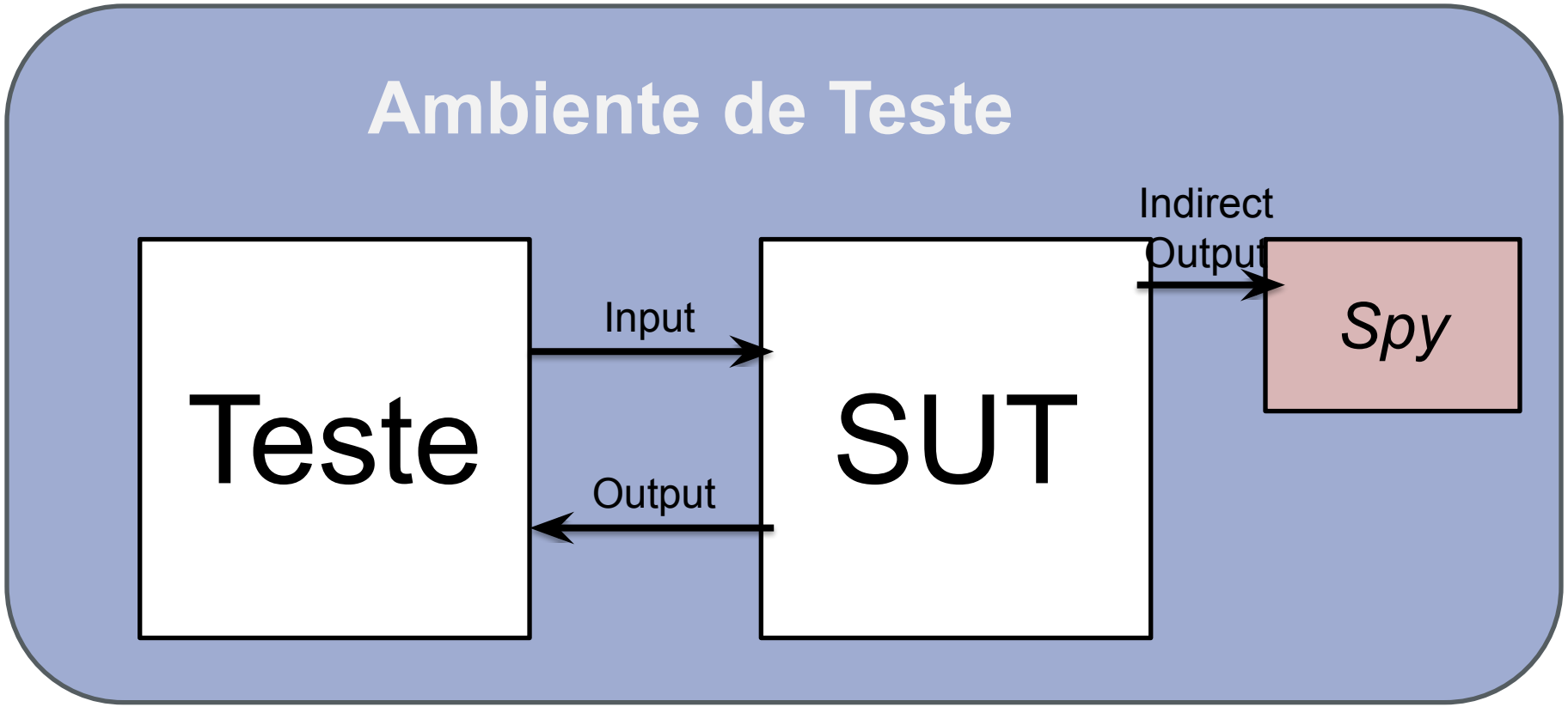
```
public static String readContent(String dir, String fileName) throws IOException {
    final Path path = Paths.get(dir, fileName);

    try (       BufferedReader reader =
                new BufferedReader(new InputStreamReader(Files.newInputStream(path))) {
                  @Override
                  public String readLine() throws IOException {
                        throw new IOException("Sabotage for testing purposes!");
                  }
            };
            ){
        final StringBuilder builder = new StringBuilder();
        String line = null;

        while ((line = reader.readLine()) != null) {
            builder.append(line);
        }

        return builder.toString();
    } catch (IOException e) {
        LOGGER.log(Level.INFO, "Error while processing " + dir + fileName, e);
        throw e;
    }
```



Test Spy

- Objeto empregado para observar as Indirect Output e as chamadas feitas pelo SUT
- Registra todas as chamadas recebidas a partir do SUT para que possam ser checadas posteriormente

Test Spy

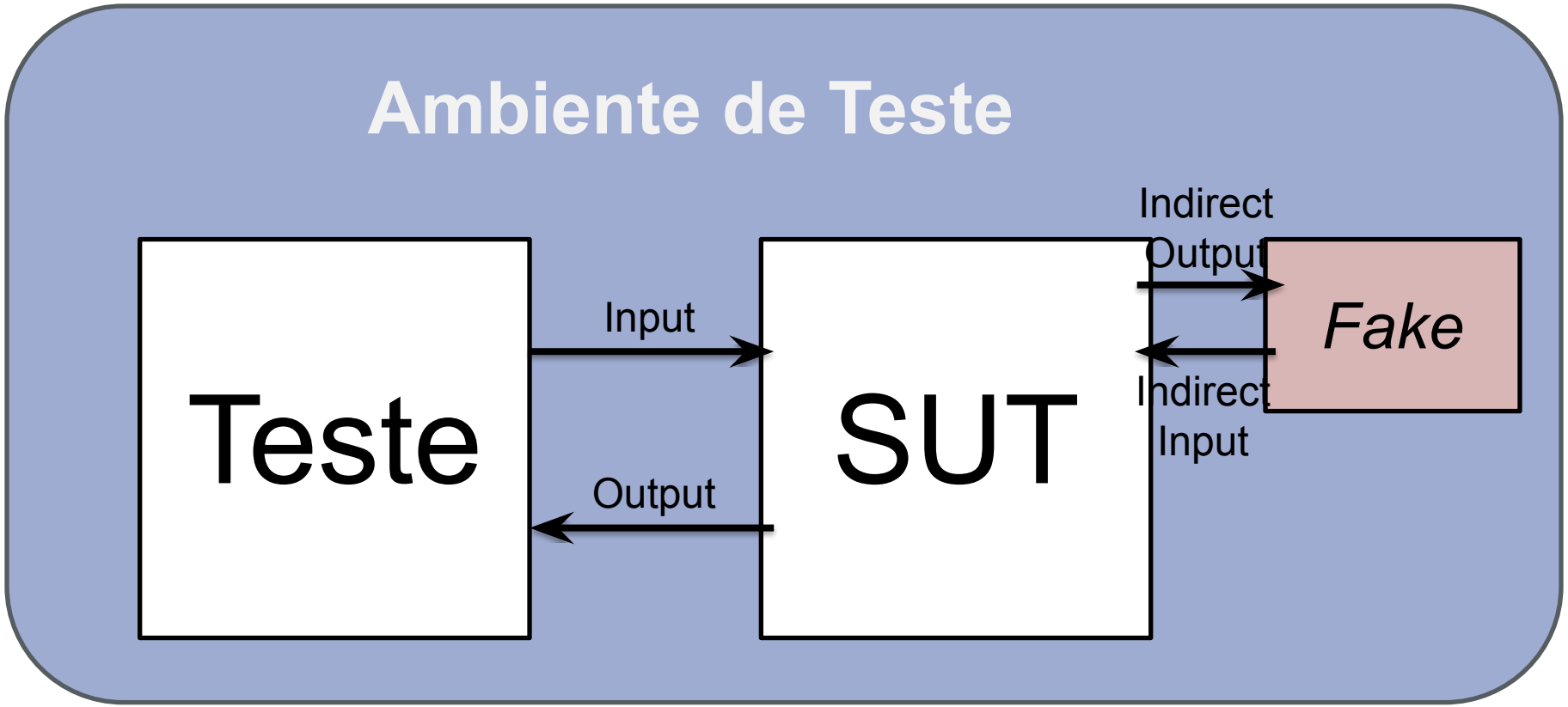

Ambiente de Teste
Indirect Output
Input
Teste
SUT
Spy
Output

```
@Test
public void testReadingCount() throws IOException {
        ReaderSpy spy = new ReaderSpy();

        readContent("/test/resources/", "355685.csv", spy);

        assertEquals(200, spy.getReadingCount());
}

public static String readContent(String dir, String fileName, final ReaderSpy spy) {
        final Path path = Paths.get(dir, fileName);

        try (
                BufferedReader reader =
                new BufferedReader(new InputStreamReader(Files.newInputStream(path))) {

                @Override
                public String readLine() throws IOException {
                        spy.increment();
                        return super.readLine();
                }
        };
        ){
                final StringBuilder builder = new StringBuilder();
```

Fake

- Objeto que não é controlado nem observado por um teste e é empregado para substituir um DOC por outras razões além de verificar Indirect Inputs e Indirect Outputs
- Tipicamente são empregados quando:
  - DOC é muito lento
  - DOC ainda não está totalmente implementado
  - DOC não está disponível no ambiente de testes

Fake


Ambiente de Teste
Teste
Input
Output
SUT
Indirect
Output
Fake
Indirect
Input

```
public class ItemService {
	private ItemRepository repository;

	public ItemService(ItemRepository repository) {
		this.repository = repository;
	}

	public void saveItem(Item item) {
		if(!checkPreConditions()) {
			throw new IllegalStateException("Pre-condition not met.");
		}

		process(item);

		if(!checkPostCondition(item)) {
			throw new IllegalStateException("Post-condition not met.");
		}

		repository.save(item);
	}
	...
}
```

```
@Test
public void testSuccessfulSaveItem(){
        ItemRepository repository = ItemRepositoryFactory.getInstance().getRepository();
        ItemService service = new ItemService(repository );

        service.saveItem(item);

        Assert.assertTrue( someCondition(item) );
        Assert.assertEquals( repository.read(item.getId()), item);
}
```

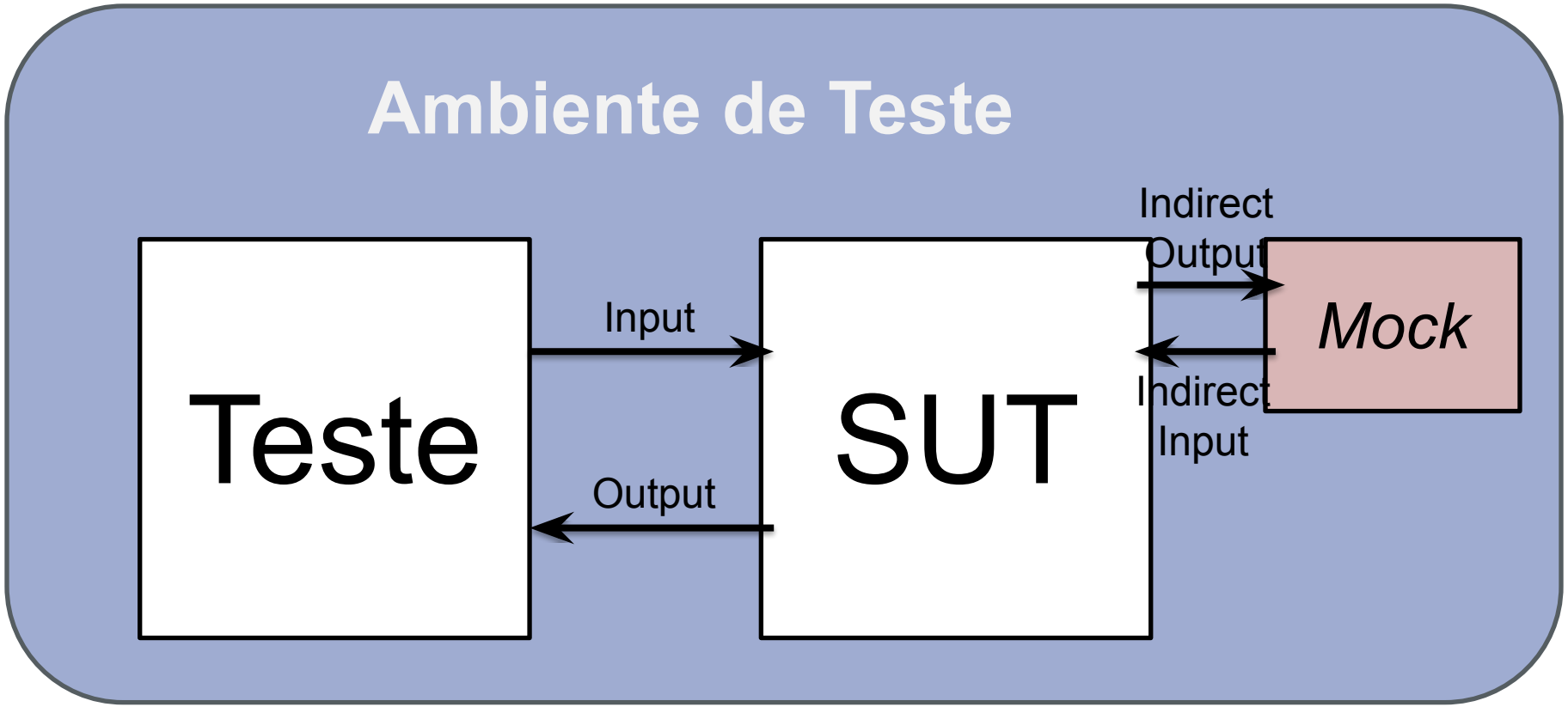
```java
public class InMemoryRepository implements ItemRepository{

	Map<BigInteger, Item> savedItems = new HashMap<>();

	@Override
	public void save(Item i) {
		savedItems.put(i.getId(), i);
	}

	@Override
	public Item read(BigInteger id) {
		return savedItems.get(id);
	}

}
```

```
@Test
public void testSuccessfulSaveItem(){
    ItemRepository repository = new InMemoryRepository();
    ItemService service = new ItemService(repository );

    service.saveItem(item);

    Assert.assertTrue( someCondition(item) );
    Assert.assertEquals( repository.read(item.getId()), item);
}
```

Mock Object

- Objeto empregado para observar as Indirect Output e as chamadas feitas pelo SUT, além de também passar Indirect Inputs para o SUT, quando necessário
- Enquanto o Spy apenas checa a quantidade de chamadas e o estado no fim do teste, o Mock faz sua checagem durante a execução do teste
  - Se durante a execução do teste alguma expectativa configurada no mock object é quebrada, o teste falha

Mock Object


Ambiente de Teste
Teste
Input
Output
SUT
Indirect
Output
Mock
Indirect
Input

Mock Object

- Para compreender melhor o funcionamento dos Mock Objects, faça o tutorial de introdução ao framework Mockito disponível no SIGAA

Referências

- MESZAROS, Gerard. xUnit test patterns: Refactoring test code. Pearson Education, 2007

- http://xunitpatterns.com/

Residência TRF5
Gerência de Configuração e Teste de Software

Prof. Eiji Adachi