



Curso Java Básico

Uma introdução prática usando
BlueJ



Comportamento mais sofisticado

Utilizando classes de biblioteca para
implementar uma funcionalidade
mais avançada

Principais conceitos a serem abordados

- Utilizando classes de biblioteca
- Pacotes e importação
- Lendo a documentação
- Escrevendo a documentação
- Mapas, listas, conjuntos
- Ocultamento de informações

A biblioteca da classe Java

- Milhares de classes
- Milhares de métodos
- Muitas classes úteis que tornam a vida muito mais fácil
- Um programador Java competente precisa ser capaz de trabalhar com bibliotecas

Trabalhando com a biblioteca

Você deve:

- conhecer algumas classes importantes pelo nome; e
- saber como descobrir outras classes.

Lembre-se:

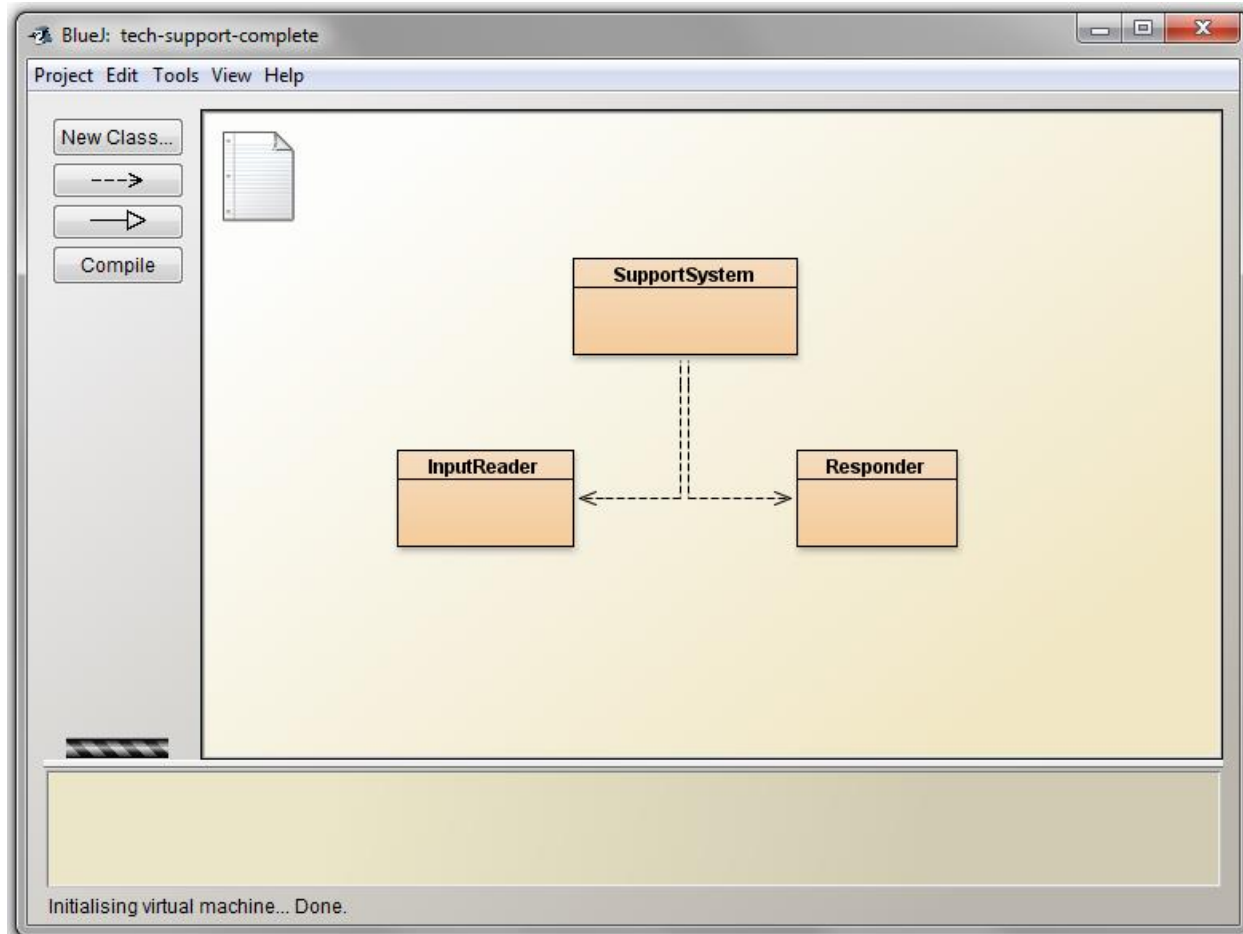
- precisamos conhecer apenas a interface, não a implementação.

Um sistema de suporte técnico

- Um sistema de diálogos textuais.
- Simula o atendimento da DodgySoft aos clientes com problemas técnicos.

(projeto baseada no '*Eliza*', desenvolvido por Joseph Weizenbaum no MIT, anos 60)

Um sistema de suporte técnico



Exercício

- **Sistema de suporte técnico**

- Inicie o **BlueJ**, abra o projeto *tech-support-complete* e crie uma instância de *SupportSystem*.
- Chame *start* e solicite ajuda para resolver um problema técnico que você poderia ter com um software da DodgySoft. O propósito do exercício é dar uma ideia do que planejamos alcançar.
- Faça o mesmo usando o projeto *tech-support1*. Qual é o comportamento observado ?

Código-fonte: InputReader

```
public class InputReader
{
    private Scanner reader;

    public InputReader() {
        reader = new Scanner(System.in);
    }

    public String getInput() {
        System.out.print("> ");           // print prompt
        String inputLine = reader.nextLine();
        return inputLine;
    }
}
```

Código-fonte: Responder

```
public class Responder
{
    public Responder()
    {
    }
    public String generateResponse()
    {
        return "That sounds interesting. Tell me more...";
    }
}
```

Exercício

- **Sistema de suporte técnico**
 - Crie uma instância de *InputReader* e explore o método *getInput*.
 - Crie uma instância de *Responder* e explore o método *generateResponse*.

Código-fonte: SupportSystem

```
public class SupportSystem
{
    private InputReader reader;
    private Responder responder;

    public SupportSystem()
    {
        reader = new InputReader();
        responder = new Responder();
    }
}
```

Código-fonte: SupportSystem

```
private void printWelcome()  
{  
    System.out.println("Welcome to the DodgySoft...");  
    System.out.println();  
    System.out.println("Please tell us ...");  
    System.out.println("We will assist you ...");  
    System.out.println("Please type 'bye' to exit..");  
}  
  
private void printGoodbye()  
{  
    System.out.println("Nice talking to you. Bye...");  
}
```

Código-fonte: SupportSystem

```
public void start()
{
    printWelcome();
    boolean finished = false;
    while(!finished) {
        String input = reader.getInput();
        if(input.startsWith("bye")) {
            finished = true;
        }
        else {
            String response = responder.generateResponse();
            System.out.println(response);
        }
    }
    printGoodbye();
}
```

Obtém entrada

Testa condição de saída

Envia resposta

Obtém resposta

Estrutura do loop principal

```
boolean finished = false;
```

```
while(!finished) {
```

```
    (obtém a entrada)
```

```
    if(condição de saída) {
```

```
        finished = true;
```

```
    }
```

```
    else {
```

```
        (obtém e envia a resposta)
```

```
    }
```

```
}
```

Refatoração

- Refatoração (do inglês *Refactoring*) é uma técnica disciplinada de reestruturar um código, alterando sua estrutura interna sem alterar seu comportamento externo.
- O uso desta técnica aprimora a concepção (*design*) do *software*, tornando-o mais legível e facilitando sua manutenção.

Refatoração

- Testar o *software* antes e depois da refatoração é fundamental para garantir que o comportamento externo não foi alterado.
- Testes automatizados favorecem a prática rotineira da refatoração.

Refatoração

- Kent Beck, criador da Programação Extrema, afirma que refatoração deve ser utilizada quando o "código cheirar mal" (*bad smells in code*).
- Alguns indícios amplamente aceitos de *bad smells* :
 - código duplicado
 - método longo
 - classe grande
 - lista de parâmetros longa

Refatoração

<http://www.refactoring.com/catalog/>

- Encapsulate Field
- Rename Method
- Parameterize Method
- Extract Method & Extract Class
- Inline Method & Inline Class
- Move Field & Move Method
- Pull Up Field & Pull Up Method
- Push Down Field & Push Down Method

Refatoração

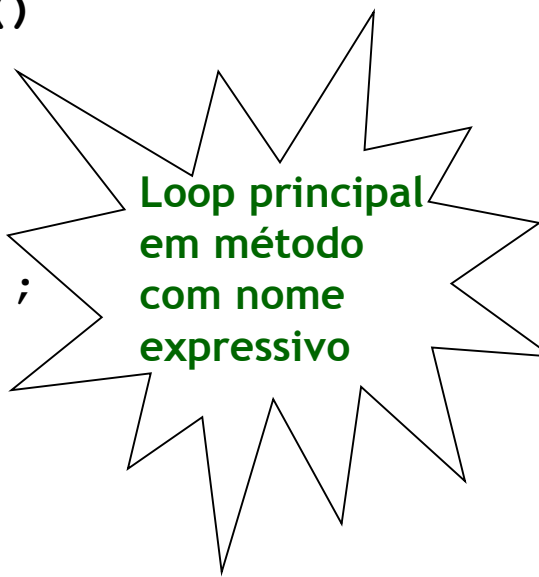
- Extract Method extrai um fragmento de código para um método cujo nome explica seu propósito.
- Usando quando:
 - um método é muito longo e cada fragmento necessita um comentário para explicar seu propósito
 - um método possui fragmentos com diferentes níveis de abstração

Código-fonte: SupportSystem

```
public void start()  
{  
    printWelcome();  
    dialogWithTheUserUntilBye();  
    printGoodbye();  
}
```

Código-fonte: SupportSystem

```
public void dialogWithTheUserUntilBye ()
{
    boolean finished = false;
    while(!finished) {
        String input = reader.getInput();
        if(input.startsWith("bye")) {
            finished = true;
        }
        else {
            String response = responder.generateResponse();
            System.out.println(response);
        }
    }
}
```



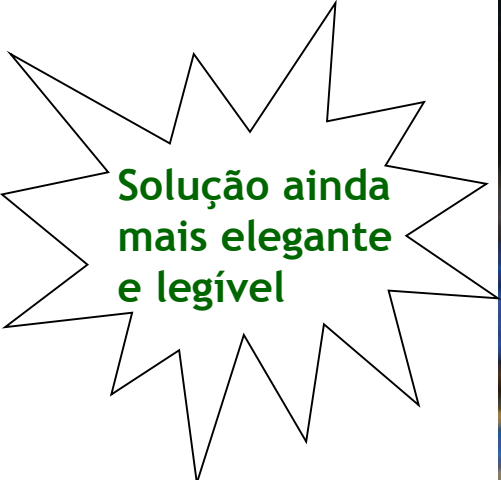
Loop principal
em método
com nome
expressivo

A instrução *break*

- A instrução `break` encerra a execução de loops `for`, `while` e `do-while`.
- Também encerra a execução de instrução `switch`.

Código-fonte: SupportSystem

```
public void dialogWithTheUserUntilBye ()
{
    while(true) {
        String input = reader.getInput();
        if(input.startsWith("bye")) {
            break;
        }
        String response = responder.generateResponse();
        System.out.println(response);
    }
}
```



Solução ainda
mais elegante
e legível

Exercício

- **Sistema de suporte técnico**
 - Extraia o loop principal de *start* para um método criado com o nome *dialogWithTheUserUntilBye*.
 - Compile a classe `SupportSystem`, crie uma instância e teste o método *start*.

Revisão (1)

- **Refatoração** é uma técnica que aprimora a concepção (*design*) do *software*, tornando-o mais legível e facilitando sua manutenção.
- Esta técnica deve ser utilizada quando o “código cheirar mal”.
- Testes automatizados favorecem o não adiamento de refatorações necessárias.

Condição de saída do loop principal

```
String input = reader.getInput();
```

```
if(input.startsWith("bye")) {  
    finished = true;  
}
```

- De onde `startsWith` vem?
- O que ele é? O que ele faz?
- Como podemos descobrir isso?

Lendo a documentação da classe

- Documentação das bibliotecas Java no formato HTML.
- Legível em um navegador Web.
- API: “*Application Programming Interface*”.
- Descrição da interface para todas as classes de biblioteca.

Interfaces *versus* implementação

A documentação inclui:

- o nome da classe;
- uma descrição geral da classe;
- uma lista dos construtores e métodos;
- valores de retorno e parâmetros para construtores e métodos; e
- uma descrição do propósito de cada construtor e método.



a interface da classe

Interfaces *versus* implementação

A documentação não inclui:

- campos privados (a maioria dos campos é privado);
- métodos privados; e
- o corpo (código-fonte) de cada método.

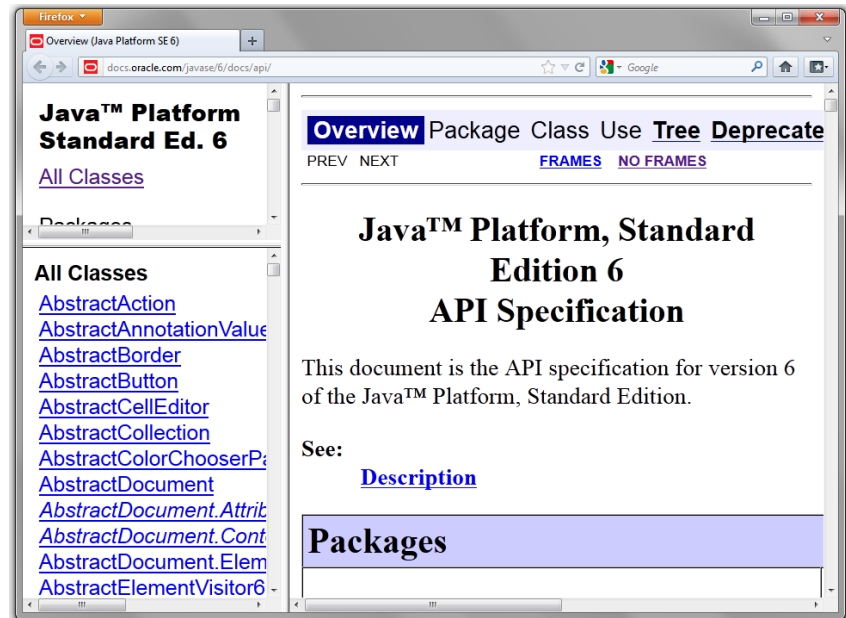
 *a implementação da classe*

Interfaces *versus* implementação

- Interface da classe
 - descreve o que uma classe faz e como ela pode ser utilizada; ela abstrai a implementação da classe.
- Implementação da classe
 - é o código-fonte completo da classe.

Lendo a documentação da API Java

- No BlueJ, usamos o item **Java Class Libraries** do menu **Help** para acessar, via navegador web, a documentação da API Java.



Exercício

- **Lendo a documentação da API**
 - Usando o item Java Class Libraries do menu Help, localize a documentação da classe *String*.
 - Localize a descrição dos métodos *startsWith*, *trim* e *toLowerCase*.

Lendo a documentação de classes parametrizadas

- A documentação mostra a previsão de *um tipo parametrizado*:
 - `ArrayList<E>`
- O nome desse tipo reaparece nos parâmetros e tipo de retorno:
 - `E get(int index)`
 - `boolean add(E e)`

Lendo a documentação de classes parametrizadas

- Um `ArrayList<TicketMachine>` realmente tem os métodos:
 - `TicketMachine get(int index)`
 - `boolean add(TicketMachine e)`

Exercício

- **Lendo a documentação da API**
 - Localize a documentação da classe *ArrayList*.
 - Localize a descrição dos métodos *add*, *get* e *contains*.

Utilizando métodos de classe de biblioteca

- Classes da biblioteca devem ser importadas utilizando uma instrução `import`
- Classes do pacote `java.lang` (usadas muito frequentemente) não precisam ser importadas; podem ser utilizadas como classes do projeto atual.

Pacotes e importação

- Classes são organizadas em pacotes.
- Podemos importar classes individuais

```
import java.util.ArrayList;  
import java.util.Random;
```
- Podemos importar o pacote completo

```
import java.util.*;
```

Exercício

- **Lendo a documentação da API**
 - Identifique na documentação quais são as classes do pacote ***java.lang***.
 - Quais já foram usadas no curso ? O que mais existe neste pacote ?

A condição de saída

```
String input = reader.getInput();
```

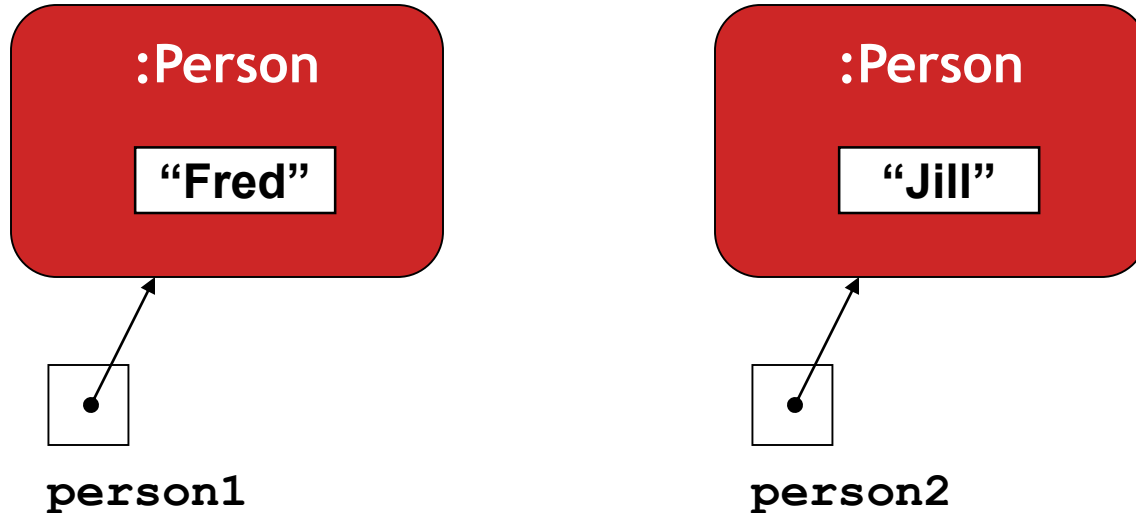
```
if(input.startsWith("bye")) {  
    finished = true;  
}
```

- Uma alternativa seria verificar se a string de entrada é a string “bye”.

```
if(input == "bye") { // funciona ?  
    ...  
}
```


Identidade versus igualdade

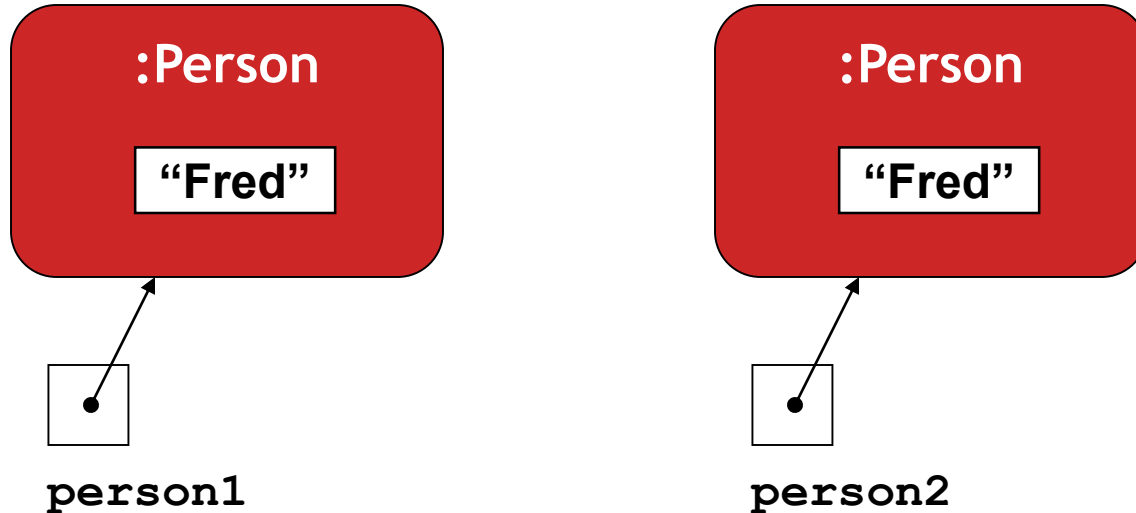
Outros objetos (não-string) :



`person1 == person2 ?`

Identidade versus igualdade

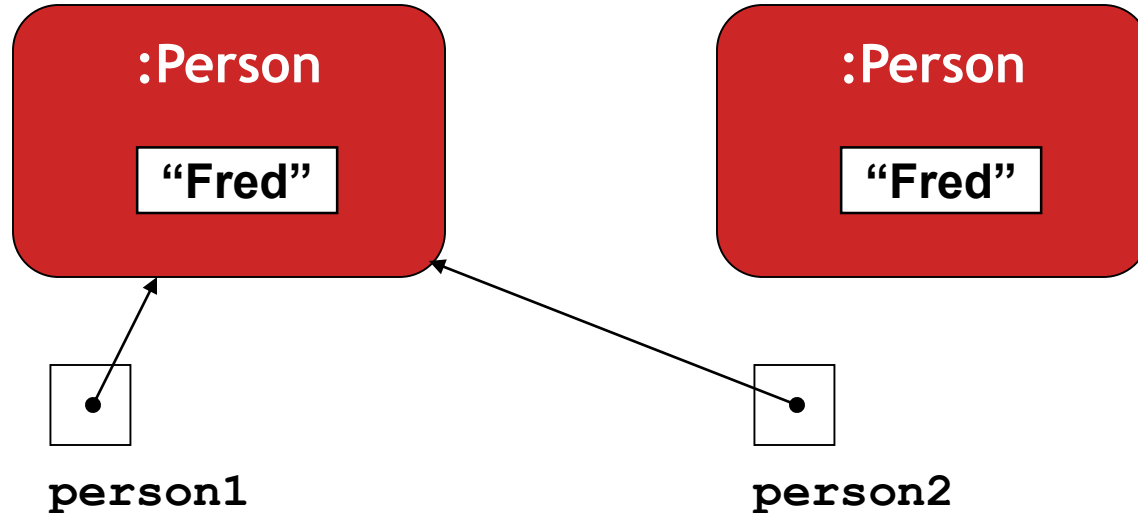
Outros objetos (não-string) :



`person1 == person2 ?`

Identidade versus igualdade

Outros objetos (não-string) :

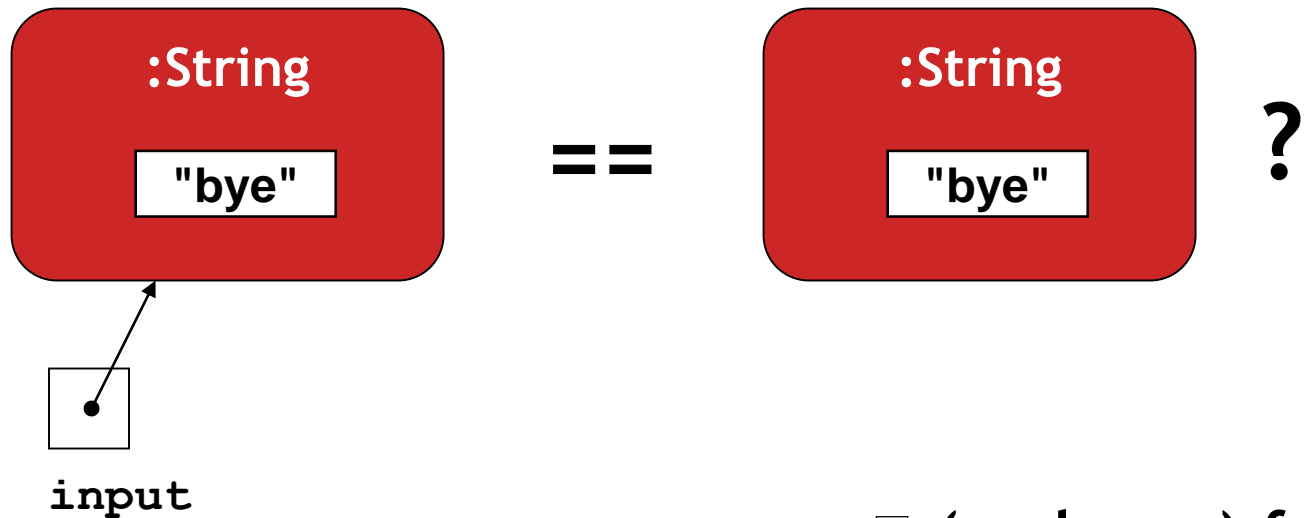


`person1 == person2 ?`

Identidade *versus* igualdade (Strings)

```
String input = reader.getInput();  
if(input == "bye") {  
    ...  
}
```

**== testa a
identidade**
(mesmo objeto)

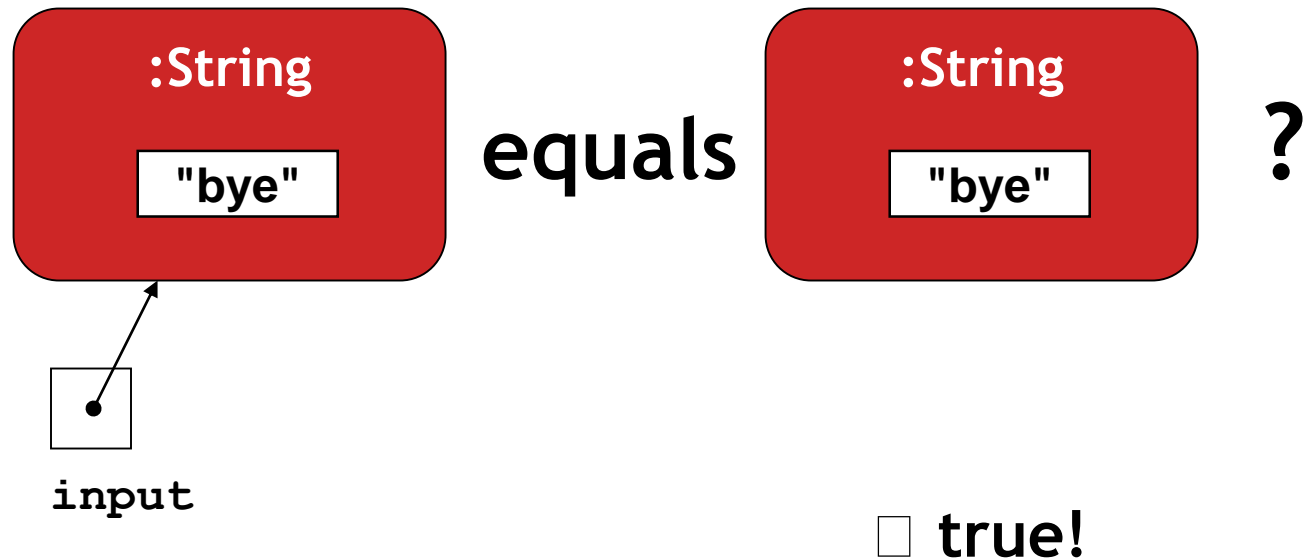


☐ (pode ser) false!

Identidade *versus* igualdade (Strings)

```
String input = reader.getInput();  
if(input.equals("bye")) {  
    ...  
}
```

**equals testa
a igualdade**
(mesmo conteúdo)



Revisão (2)

- A documentação das bibliotecas Java apresenta descrição da interface pública de suas classes.
- Classes da biblioteca (exceto `java.lang`) devem ser importadas utilizando uma instrução `import`.
- Identidade *versus* igualdade:
 - `==` testa identidade (mesmo objeto)
 - `equals` testa igualdade (mesmo conteúdo)

Exercício

- **Sistema de suporte técnico**
 - Feche o projeto anterior, abra o projeto *tech-support2* e crie uma instância de *SupportSystem*.
 - Chame *start* e solicite ajuda para resolver um problema técnico. Qual é o comportamento observado ?

Gerando respostas aleatórias

- A classe de biblioteca **Random** pode ser utilizada para gerar números aleatórios:
 1. cria-se uma instância da classe **Random**
 2. chama-se um método nesta instância

```
import java.util.Random;  
...  
Random randomGenerator = new Random();  
...  
int index1 = randomGenerator.nextInt();  
int index2 = randomGenerator.nextInt(100);
```


Exercício

- **Lendo a documentação da API**
 - Localize a documentação da classe *Random*.
 - Localize a descrição dos métodos usados para obter números pseudo-aleatórios do tipo *int* e *double*. É possível definir limites mínimo e máximo ?

Código-fonte: Responder

```
public Responder()
{
    randomGenerator = new Random(); ← Instância de Random
    responses = new ArrayList<String>(); ← Coleção de respostas
    fillResponses();
}

public String generateResponse()
{
    int index = randomGenerator.nextInt(responses.size()); ← Obtém número inteiro (de 0 a responses.size)
    return responses.get(index); ← Retorna resposta
}

public void fillResponses()
{
    ... ← Preenche coleção de respostas
}
```

Código-fonte: Responder

```
private void fillResponses()  
{  
    responses.add("That sounds odd. Could you describe that problem in more detail?");  
    responses.add("No other customer has ever complained about this before. \n" +  
        "What is your system configuration?");  
    responses.add("That's a known problem with Vista. Windows 7 is much better.");  
    responses.add("I need a bit more information on that.");  
    responses.add("Have you checked that you do not have a dll conflict?");  
    responses.add("That is explained in the manual. Have you read the manual?");  
    responses.add("Your description is a bit wishy-washy. Have you got an expert\n" +  
        "there with you who could describe this more precisely?");  
    responses.add("That's not a bug, it's a feature!");  
    responses.add("Could you elaborate on that?");  
}
```

Exercício

- **Sistema de suporte técnico**
 - Feche o projeto anterior, abra o projeto *tech-support-complete* e crie uma instância de *SupportSystem*.
 - Chame *start* e solicite ajuda para resolver um problema técnico.
 - Após algumas respostas, diga que há um bug no software. Algumas respostas depois pergunte se usar o Windows influi no problema. Qual é o comportamento observado ?

Simulando respostas sensíveis ao contexto

- Podemos selecionar um conjunto de palavras que provavelmente ocorrerá em perguntas comuns e associaremos estas palavras a respostas particulares.
 - Caso a pergunta tenha alguma palavra do conjunto, a resposta será sensível ao contexto
 - Caso contrário, a resposta continuará aleatória
- Este é um método grosseiro, mas pode ser surpreendentemente eficaz.

Simulando respostas sensíveis ao contexto

- Precisaremos:
 - associar palavras prováveis a respectivas respostas particulares
 - comparar cada palavra da pergunta com as palavras prováveis

Maps

- Maps são ‘coleções’ que armazenam pares ordenados como elementos.
- Pares ordenados consistem em uma *chave* e um *valor*.
- Pesquisam palavras fornecendo uma chave e recuperando um valor.
- Um exemplo: uma lista telefônica.

Utilizando maps

- Um `map` com strings como chaves e valores

`:HashMap`

"Charles Nguyen"	"(531) 9392 4587"
------------------	-------------------

"Lisa Jones"	"(402) 4536 4674"
--------------	-------------------

"William H. Smith"	"(998) 5488 0123"
--------------------	-------------------

Utilizando maps

```
HashMap<String, String> phoneBook =  
    new HashMap<String, String>();  
  
phoneBook.put("Charles Nguyen", "(531) 9392 4587");  
phoneBook.put("Lisa Jones", "(402) 4536 4674");  
phoneBook.put("William H. Smith", "(998) 5488 0123");  
  
String phoneNumber = phoneBook.get("Lisa Jones");  
System.out.println(number);
```

Lists

- Lists são coleções ordenadas (às vezes chamadas sequências).
- Lists podem conter elementos repetidos.
- Lists fornecem acesso por índice.

Sets

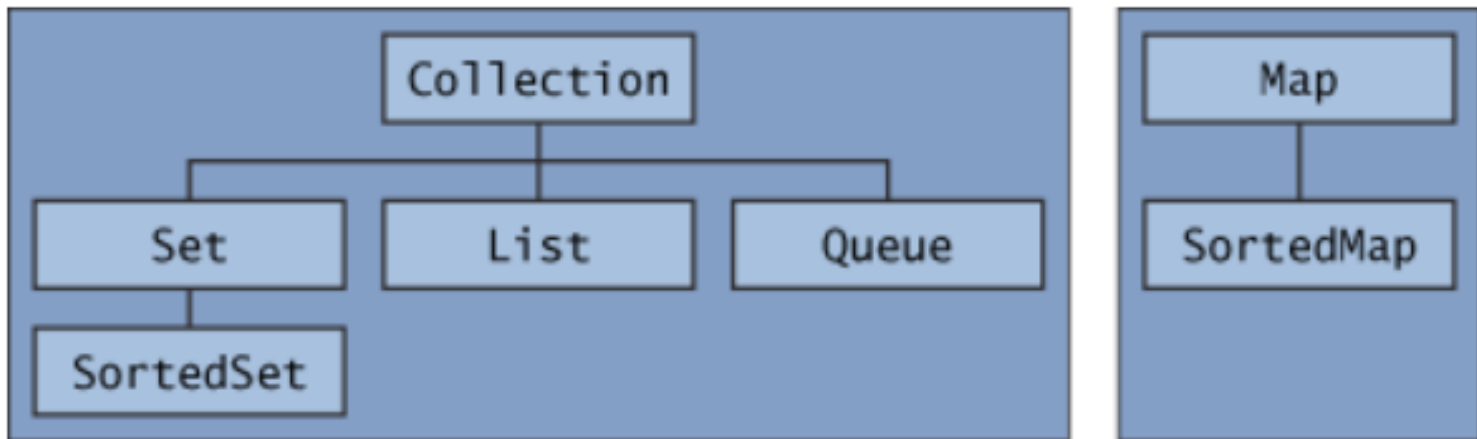
- Sets são coleções que armazenam elementos sem repetição.
- Sets não mantêm os elementos em nenhuma ordem específica.

Utilizando conjuntos

```
import java.util.HashSet;  
...  
HashSet<String> mySet = new HashSet<String>();  
  
mySet.add("one");  
mySet.add("two");  
mySet.add("three");  
  
for(String element : mySet) {  
    do something with element  
}
```

**Compare isso
com o código
de ArrayList !**

Coleções



The core collection interfaces of
Java Collections Framework

Dividindo strings

- A classe de biblioteca **String** possui o método **split** para dividir um string em um array de strings.

```
String txt = "one-two-three";  
String delimiter = "-";  
String[] wordArray;  
  
wordArray = txt.split(delimiter);  
  
for(String word : wordArray) {  
    System.out.println(word);  
}
```

Código-fonte: InputReader

```
public HashSet<String> getInput() ← Novo tipo de retorno
{
    System.out.print("> "); // print prompt
    String inputLine = reader.nextLine().trim().toLowerCase();

    String[] wordArray = inputLine.split(" "); // split at spaces

    // add words from array into hashset
    HashSet<String> words = new HashSet<String>();
    for(String word : wordArray) {
        words.add(word);
    }
    return words;
}
```

← Pode haver repetições

← Sem repetições

Código-fonte: Responder

```
public Responder() {  
    randomGenerator = new Random();  
    responseMap = new HashMap<String,String>();  
    defaultResponses = new ArrayList<String>();  
    fillResponseMap();  
    fillDefaultResponses();  
}
```

Nova lista de parâmetros



```
public String generateResponse(HashSet<String> words) {  
    for (String word : words) {  
        String response = responseMap.get(word);  
        if(response != null) {  
            return response;  
        }  
    }  
    return pickDefaultResponse();  
}
```


Código-fonte: Responder

```
private void fillResponseMap ()
{
    responseMap.put("bug",
        "Well, you know, all software has some bugs. But our software\n" +
        "engineer are working very hard to fix them. Can you describe\n" +
        "the problem a bit further?");
    responseMap.put("windows",
        "This is a known bug to do with the Windows operating system.\n" +
        "Please report it to Microsoft. There is nothing we can do\n" +
        "about this.");
    ...
}
```

Código-fonte: SupportSystem

```
public class SupportSystem
{
    private InputReader reader;
    private Responder responder;

    public SupportSystem() {
        reader = new InputReader();
        responder = new Responder();
    }

    public void start() {
        printWelcome();
        dialogWithTheUserUntilBye();
        printGoodbye();
    }
}
```

Código-fonte: SupportSystem

```
public void dialogWithTheUserUntilBye ()
{
    while(true) {
        HashSet<String> input = reader.getInput();
        if(input.contains("bye")) {
            break;
        }
        String response = responder
                        .generateResponse(input);
        System.out.println(response);
    }
}
```

Novo tipo de retorno
↓

Condição de saída com método apropriado
↙

Nova lista de parâmetros
↓

Exercício

- **Sistema de suporte técnico**
 - Quais classes precisam ser alteradas para acrescentar mais respostas particulares em nossa simulação ?
 - Acrescente algumas respostas particulares em nossa simulação e verifique se funcionam.

Escrevendo a documentação da classe

- Suas classes devem ser documentadas da mesma maneira que as classes de biblioteca.
- Outras pessoas devem ser capazes de utilizar sua classe sem precisar ler a implementação.
- Torne sua classe uma ‘classe de biblioteca’!

Elementos da documentação

A documentação de uma classe deve incluir:

- o nome da classe;
- um comentário descrevendo o propósito geral e as características da classe;
- um número da versão;
- o nome do autor (ou autores); e
- a documentação para cada construtor e cada método.

Elementos da documentação

A documentação de cada construtor e método deve incluir:

- o nome do método;
- o tipo de retorno;
- o nome e os tipos de parâmetros;
- uma descrição do propósito e da função do método;
- uma descrição de cada parâmetro; e
- uma descrição do valor retornado.

javadoc

Comentário da classe :

```
/**  
 * A classe Responder representa um objeto gerador  
 * de respostas. Ele é utilizado para gerar uma  
 * resposta automática.  
 *  
 * @author      Michael Kölling e David J. Barnes  
 * @version     1.0   (1.Feb.2002)  
 */
```

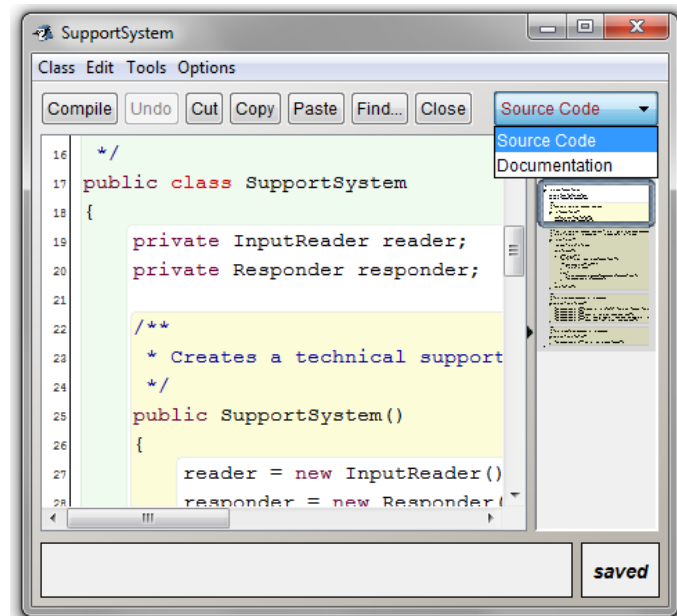
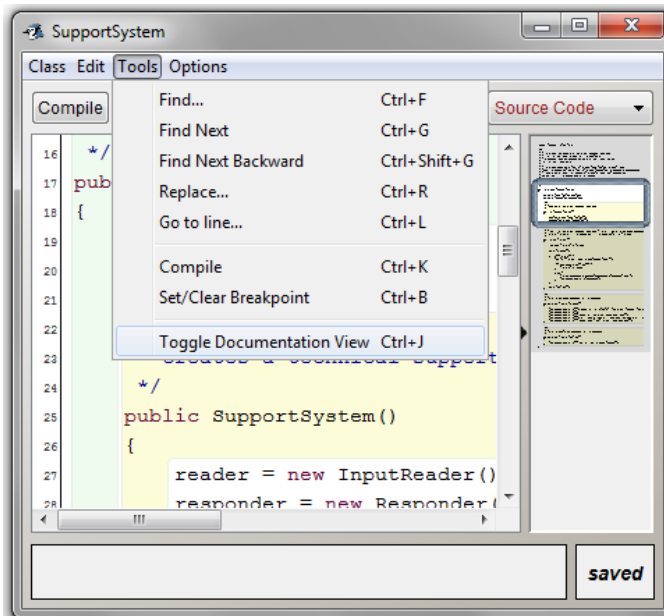

javadoc

Comentário do método:

```
/**
 * Lê uma linha de texto da entrada-padrão (o terminal
 * de texto) e a retorna como um conjunto de palavras.
 *
 * @param  prompt  Um prompt para imprimir na tela.
 * @return  Um conjunto de strings, em que cada string
 *          é uma das palavras digitadas pelo usuário.
 */
public HashSet getInput(String prompt)
{
    ...
}
```

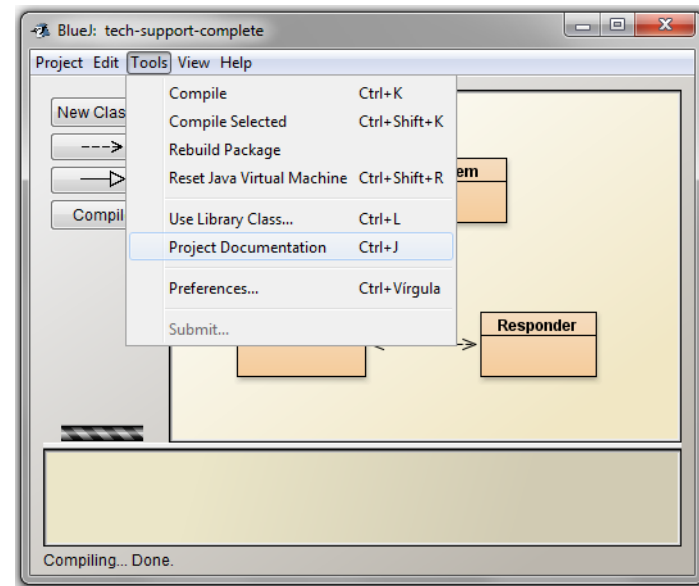
Escrevendo a documentação da classe

- No BlueJ, podemos gerar e visualizar a documentação de uma classe usando a visão Documentation do editor.



Escrevendo a documentação da classe

- Podemos gerar e visualizar a documentação de todas as classes do projeto usando a opção Project Documentation do menu Tools.



Exercício

- **Sistema de suporte técnico**
 - Gere a documentação javadoc para as classes do projeto *tech-support-complete*.
 - Localize a documentação na pasta *doc* do projeto. Chame o navegador padrão usando o arquivo *index.html* da pasta.

Ocultamento de informações

- Conhecendo a diferença entre a interface e a implementação de uma classe, podemos entender o propósito dos modificadores de acesso.
- O usuário de uma classe não deve ter permissão para conhecer sua implementação (ou, ao menos, para usar esse conhecimento). Esse princípio é chamado de *ocultamento de informações*.

Ocultamento de informações

- Dados que pertencem a um objeto são ocultados de outros objetos.
- Sabem *o que* um objeto pode fazer, não *como* ele faz isso.
- Ocultamento de informações aumenta o nível de *independência*. Isto é importante para grandes sistemas e para manutenção.
- Alterações em uma classe não deveriam provocar alterações em outras classes (acoplamento). Acoplamento fraco é bom.

Público *versus* privado

- Membros públicos (campos, métodos) são acessíveis a outras classes.
- Membros privados são acessíveis apenas dentro da mesma classe.
- Campos não devem ser públicos.
- Somente os métodos concebidos para outras classes devem ser públicos.

Público *versus* privado

- Campos não devem ser públicos.
 - permite que o objeto mantenha controle sobre seu estado
 - alteração dos campos através de métodos modificadores assegura que eles não serão configurados com valores que deixariam o objeto em um estado inconsistente

Público *versus* privado

- Somente os métodos concebidos para outras classes devem ser públicos.
 - métodos públicos fornecem operações para os usuários da classe
 - métodos privados são usados no interior da classe para:
 - dividir uma tarefa maior em tarefas menores (melhorar legibilidade do código)
 - conter tarefas usadas em diversos métodos (evitar duplicidade de código)

Código-fonte: SupportSystem

```
public class SupportSystem
{
    private InputReader reader;
    private Responder responder;

    public SupportSystem() { ... }

    public void start() { ... }
    private void printWelcome() { ... }
    private void printGoodbye() { ... }
}
```

Revisão (3)

- Java tem uma extensa biblioteca de classes.
- Um bom programador precisa conhecer a biblioteca.
- A documentação informa o que precisamos saber para utilizar uma classe (interface).
- A implementação é ocultada (ocultamento de informações).
- Podemos documentar nossas classes da mesma forma que as classes da API Java.



Contatos

Câmara dos Deputados
CENIN - Centro de Informática

Carlos Renato S. Ramos

carlosrenato.ramos@camara.gov.br