Entrada e Saída no Console

Para facilitar a entrada de dados em Java é necessário criar um objeto da classe `Scanner`, passando o `System.in`. Na saída de dados, é possível usar os métodos `print` ou `println` do `System.out`.

```
import java.util.Scanner;
public class Program {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Nome: ");
        String nome = console.nextLine();
        System.out.println("Olá, " + nome + "!");
    }
}
```

Entrada e Saída Gráficas

Você pode fazer a entrada e saída de dados por meio da interface gráfica. Note que, se você pressionar o botão "Cancelar" na janela de entrada de dados, o valor retornado será `null`.

```
import javax.swing.JOptionPane;
public class Program {
    public static void main(String[] args) {
        String nome = JOptionPane.showInputDialog(null, "Nome");
        JOptionPane.showMessageDialog(null, "Olá, " + nome + "!");
    }
}
```

Strings Formatadas

Na saída de dados, também é possível usar o método `printf` para a criação de strings formatadas sem o uso de concatenação. Use `%s` para strings, `%f` para doubles, `%.2f` para doubles com duas casas decimais e `%d` para inteiros. Não esqueça do `\n` no final caso queira pular uma linha (`%n` tem o mesmo efeito, mas é independente de sistema operacional). O uso de `String.format` é similar, mas somente cria a string, sem mostrá-la na tela. Para maiores informações, leia o JavaDoc: http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html

```
import javax.swing.JOptionPane;
public class Program {
    public static void main(String[] args) {
        String nome = JOptionPane.showInputDialog("Nome");
        System.out.printf("Olá, %s!", nome);
        String formatada = String.format("Olá, %s!", nome);
        JOptionPane.showMessageDialog(null, formatada);
    }
}
```

Conversão de Tipos

Para converter de string para um tipo numérico, usa-se:

```
int i = Integer.parseInt(console.nextLine());
double d = Double.parseDouble(console.nextLine());
...
```

Para converter de um número para string usa-se:

```
String s = String.valueOf(numero);
```

Comparação de Objetos e Strings

A comparação de objetos deve ser feita com o método `equals`, pois o operador `==` compara as referências! Como, em Java, as strings são objetos da classe `String`, elas devem ser comparadas dessa mesma forma:

```
import java.util.Scanner;
public class Program {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Senha: ");
        String senha = console.nextLine();
        if (senha.equals("Java")) {
          System.out.println("Senha certa!");
        } else {
          System.out.println("Senha errada!");
        }
    }
}
```

Números Aleatórios

Para sortear números, precisamos criar um objeto da classe `Random` e usar o método `nextInt(n)`, que retorna um número aleatório entre 0 e n-1.

```
import java.util.Random;
public class Program {
    public static void main(String[] args) {
        Random rng = new Random();
        int numero = rng.nextInt(10);
        System.out.printf("Número sorteado = %d", numero);
    }
}
```

Vetores e Listas

Para criar vetores, a sintaxe é similar à do C#.

```
import java.util.Scanner;
public class Program {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        int[] vetor = new int[10];
        for (int i = 0; i < vetor.length; i++) {
            System.out.printf("Entre com o %dº elemento: ", i + 1);
            int n = Integer.parseInt(console.nextLine());
            vetor[i] = n;
        }
        for (int i = 0; i < vetor.length; i++) {
            int n = vetor[i];
            System.out.println(n);
        }
    }
}
```

```
// ou, usando o equivalente ao foreach do C#
for (int n : vetor) {

    System.out.println(n);
}
```

As listas são criadas usando a interface `List` e alguma classe que implementa essa interface. A mais usada é a `ArrayList`. Note que as listas armazenam apenas objetos. Assim, se quisermos armazenar `int`s, devemos indicar a classe equivalente que é `Integer`. Idem para `double`s e outros tipos.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class Program {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        List<Integer> lista = new ArrayList<Integer>();
        for (int i = 0; i < 10; i++) {
            System.out.printf("Entre com o %dº elemento: ", i + 1);
            int n = Integer.parseInt(console.nextLine());
            lista.add(n);
        }
        for (int i = 0; i < lista.size(); i++) {
            int n = lista.get(i);
            System.out.println(n);
        }
    }
}
```

```
// ou, usando o equivalente ao foreach do C#
for (int n : lista) {

    System.out.println(n);
}
```

Tratamento de Erros

O tratamento de erros é similar ao do C#. As maiores diferenças estão nos nomes das exceções, que são diferentes, e no bloco `catch`, que deve sempre declarar a variável que irá receber a exceção.

```
import java.util.Scanner;
public class Program {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        try {
            System.out.print("Entre com um número inteiro maior ou igual à zero: ");
            int n = Integer.parseInt(console.nextLine());
            if (n < 0) {
                throw new IllegalArgumentException();
            }
            System.out.println("O número digitado foi: " + n);
        } catch (NumberFormatException ex) {
            System.out.println("Você não entrou com um número inteiro!");
        } catch (IllegalArgumentException ex) {
            System.out.println("O número digitado não é maior ou igual à zero!");
        }
    }
}
```

Ordenação de Vetores e Listas

Para ordenar vetores, usamos o comando `Arrays.sort(v)` e, para ordenar listas, usamos o comando `Collections.sort(l)`. A situação-problema é quando queremos ordenar objetos de uma classe criada por nós. Nesse caso, como o algoritmo vai saber qual objeto deve vir antes do outro? Para resolver isso, a classe deve implementar a interface `Comparable`, que contém o método abstrato `int compareTo(other)`. Ao implementar este método, devemos retornar um número negativo caso o objeto `this` deva vir antes de `other`, retornar um número positivo caso o objeto `this` deva ir após `other` e retornar zero caso eles sejam iguais. Exemplo:

```
public class Aluno implements Comparable<Aluno> {
    private String nome;
    private double p1, p2;
    public Aluno(String nome, double p1, double p2) {
        this.nome = nome;
        this.p1 = p1;
        this.p2 = p2;
    }
    public String getNome() { return nome; }
    public double getMedia() { return (p1 + p2) / 2; }
    public boolean getPassou() { return getMedia() >= 6; }
    @Override
    public int compareTo(Aluno other) {
        return this.nome.compareTo(other.nome); // a classe String já implementa Comparable; nesse caso, iremos ordenar por nome.
    }
}
```

```
import java.util.Arrays;


public class Program {
    public static void main(String[] args) {
        Aluno[] alunos = new Aluno[3];
        alunos[0] = new Aluno("Carlos", 3, 6);
        alunos[1] = new Aluno("Amanda", 7, 8);
        alunos[2] = new Aluno("Bianca", 4, 9);

        Arrays.sort(alunos);

        for (Aluno aluno : alunos) {
            System.out.println(aluno.getNome());
        }
    }
}
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class Program {
    public static void main(String[] args) {
                List<Aluno> alunos = new ArrayList<>();
        alunos.add(new Aluno("Carlos", 3, 6));
        alunos.add(new Aluno("Amanda", 7, 8));
        alunos.add(new Aluno("Bianca", 4, 9));

        Collections.sort(alunos);

        for (Aluno aluno : alunos) {
                System.out.println(aluno.getNome());
        }
    }
}
```

Quando precisamos ordenar por outros critérios, deve-se criar uma classe auxiliar que implementa a interface `Comparator`, que contém o método abstrato `int compare(o1, o2)`. O retorno deste método segue a mesma regra do método `compareTo` só que entre os objetos `o1` e `o2`. Para usar a classe criada, passamos um objeto dela para o método `sort` da classe `Arrays` ou `Collections`. Exemplo:

```
import java.util.Comparator;
public class OrdenaPorMedia implements Comparator<Aluno> {
    @Override
    public int compare(Aluno o1, Aluno o2) {
        if (o1.getMedia() > o2.getMedia()) { return 1; }
        else if (o1.getMedia() < o2.getMedia()) { return -1; }
        else { return 0; }
    }
}
```

```
import java.util.Arrays;


public class Program {
    public static void main(String[] args) {
        Aluno[] alunos = new Aluno[3];
        alunos[0] = new Aluno("Carlos", 3, 6);
        alunos[1] = new Aluno("Amanda", 7, 8);
        alunos[2] = new Aluno("Bianca", 4, 9);

        Arrays.sort(alunos, new OrdenaPorMedia());

        for (Aluno aluno : alunos) {
            System.out.println(aluno.getNome());
        }
    }
}
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class Program {
    public static void main(String[] args) {
                List<Aluno> alunos = new ArrayList<>();
        alunos.add(new Aluno("Carlos", 3, 6));
        alunos.add(new Aluno("Amanda", 7, 8));
        alunos.add(new Aluno("Bianca", 4, 9));

        Collections.sort(alunos, new OrdenaPorMedia());

        for (Aluno aluno : alunos) {
                System.out.println(aluno.getNome());
        }
    }
}
```

Tratamento de Erros Avançado

Na maior parte das vezes, é interessante criarmos nossas próprias exceções de forma a especificar melhor qual o tipo de erro que estamos tratando. Para fazer isso, devemos criar uma classe que herda de `Exception` para cada erro que queremos tratar. Vale notar que é uma classe comum e, portanto, pode ter seus próprios atributos e métodos. Exemplo:

```
public class NumeroNegativoException extends Exception {
    private int n;
    public NumeroNegativoException(int n) {
        this.n = n;
    }
    public int getNumero() { return n; }
}
```

Uma vez criada uma classe que herda (é-uma) `Exception`, podemos usá-la com o comando `throw`:

```
import java.util.Scanner;
public class Program {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        try {
            System.out.print("Entre com um número inteiro maior ou igual à zero: ");
            int n = Integer.parseInt(console.nextLine());
            if (n < 0) {
                throw new NumeroNegativoException(n);
            }
            System.out.println("O número digitado foi: " + n);
        } catch (NumberFormatException ex) {
            System.out.println("Você não entrou com um número inteiro!");
        } catch (NumeroNegativoException ex) {
            System.out.println("O número digitado (" + ex.getNumero() + ") não é maior ou igual à zero!");
        }
    }
}
```

Um cuidado que devemos ter é quando geramos exceções dentro de métodos e/ou construtores. Quando não os tratamos (com `try-catch`), devemos colocar o comando `throws` na declaração. Exemplo:

```
public class Fatorial {
    private int n;
    public Fatorial(int n) throws NumeroNegativoException {
        setN(n); // precisa do throws porque o método setN gera a exceção e não estamos tratando-a aqui
    }
    public void setN(int n) throws NumeroNegativoException {
        if (n < 0) {
            throw new NumeroNegativoException(n);
        }
        this.n = n;
    }
    public int getResultado() {
        int f = 1;
        for (int i = 1; i <= n; i++) {
            f *= i;
        }
        return f;
    }
}
```

Outra situação bem comum é lançar uma exceção encapsulando uma outra exceção, causadora do erro. Exemplo:

```
public class MinhaExcecao extends Excecao {
    public MinhaExcecao(Throwable cause) {
        super(cause);
    }
}

public class UmaClasse {
    public void umMetodo() throws MinhaExcecao {
        try {
            ... // código que pode gerar uma exceção da classe ExcecaoCausa
        }
        catch (ExcecaoCausa ex) {
            throw new MinhaExcecao(ex);
        }
    }
}
```

Operações com Strings

A classe `String` possui alguns métodos bastante úteis. Segue alguns deles:

`s.charAt(n)`	retorna o caractere localizado na posição `n` da string `s`
`s.startsWith("texto")`	verfica se a string `s` inicia com o texto passado
`s.endsWith("texto")`	verfica se a string `s` termina com o texto passado
`s.equals("texto")`	compara a string `s` com o texto passado, caractere por caractere
`s.equalsIgnoreCase("texto")`	compara a string `s` com o texto passado, caractere por caractere, ignorando se é maiúsculo / minúsculo
`s.indexOf("texto")`	retorna a posição da primeira ocorrência na string `s` do texto passado
`s.lastIndexOf("texto")`	retorna a posição da última ocorrência na string `s` do texto passado
`s.length()`	retorna o tamanho da string `s`
`s.substring(i, f)`	retorna um pedaço da string `s`, começando na posição `i` e terminando na posição `f`
`s.substring(i)`	retorna um pedaço da string `s`, começando na posição `i` e indo até o final
`s.toLowerCase()`	retorna a string `s` toda em letras minúsculas
`s.toUpperCase()`	retorna a string `s` toda em letras maiúsculas
`s.trim()`	retorna a string `s` sem espaços à sua volta
`s.isEmpty()`	verifica se a string `s` está vazia (`""`)
`String.valueOf(...)`	retorna uma string equivalente ao valor fornecido (conversão de `int` para string, por exemplo)

Operações Matemática

A classe `Math` possui alguns métodos bastante úteis. Segue alguns deles:

`Math.sqrt(n)`	Calcula a raiz de n
`Math.pow(x, y)`	Eleva x a y (x^y)
`Math.PI`	Valor de Pi (π)

Atalhos de IDEs

Eclipse	NetBeans	Resultado
Ctrl-Espaço	Ctrl-Espaço	Auto-Completar
Ctrl-Shit-O	Ctrl-Shift-I	Arruma imports
Ctrl-Shift-F	Alt-Shift-F	Formata o código (desde que não haja erros de sintaxe)
Ctrl-1	Alt-Enter	Menu de correção de erros
Alt-Shift-S	Alt-Insert	Menu de geração de código (construtor, getters e setters, implementar método, etc.)
Alt-Shift-T	Botão-Direito > Refatorar	Menu de refatorações
Alt-Shit-R	Ctrl-R	Renomear variável, classe, método, etc.
Alt-Shift-M	Alt-Shift-M	Introduzir método
main<Ctrl-Espaço>	psvm<Tab>	`public static void main(String[] args) { }`
syso<Ctrl-Espaço>	sout<Tab>	`System.out.println("");`
Alt-↑	Alt-Shift-↑	Move linha para cima
Alt-↓	Alt-Shift-↓	Move linha para baixo
Ctrl-Alt-↑	Ctrl-Shift-↑	Duplica linha para cima
Ctrl-Alt-↓	Ctrl-Shift-↓	Duplica linha para baixo
Ctrl-D	Ctrl-E	Apaga linha
Ctrl-/	Ctrl-/	Comenta/Descomenta linhas selecionadas
Ctrl-Shift-T	Ctrl-O	Vai para uma classe
Ctrl-Shift-R	Alt-Shift-O	Vai para um arquivo
F3	Ctrl-Shift-B	Vai para o a definição do tipo/classe