1 - Criar um projeto no Eclipse

Para codificar os nossos algoritmos iremos utilizar a linguagem Java e o IDE (Ambiente de Desenvolvimento Integrado) Eclipse.

Crie um novo projeto no Eclipse acessando File -> New -> Java Project (Figura 1) e na sequência forneça o nome do projeto e local onde ele será salvo no seu computador (Figura 2), clique em Finalizar para concluir o processo de criação do projeto.

O projeto criado terá a estrutura mostrada na Figura 3, o nosso código precisa estar dentro do pacote (pasta) de código fonte src.

Todo projeto Java precisa ter uma método main, que é o 1º método a ser executado no projeto. Então crie uma classe de nome Principal clicando com o botão direito do mouse sobre o pacote src e acessando New -> Class, na janela da Figura 4 forneça o nome da classe, que é Principal, e selecione a opção para o Eclipse colocar o método main nesta classe. Para melhor organizar o código forneça o pacote aula, como este pacote não existe, então o Eclipse irá criar ele também.

Após finalizar a criação da classe Principal o projeto terá a estrutura mostrada na Figura 5.

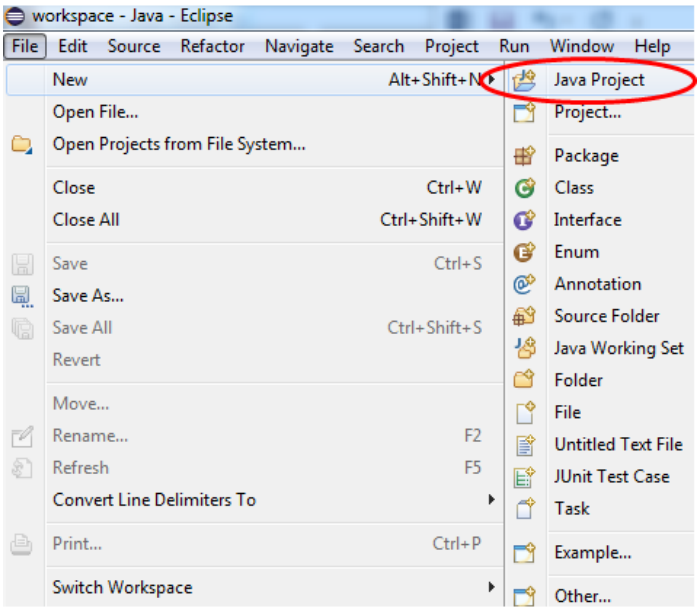

Figura 1 – Menu para criar um novo projeto.

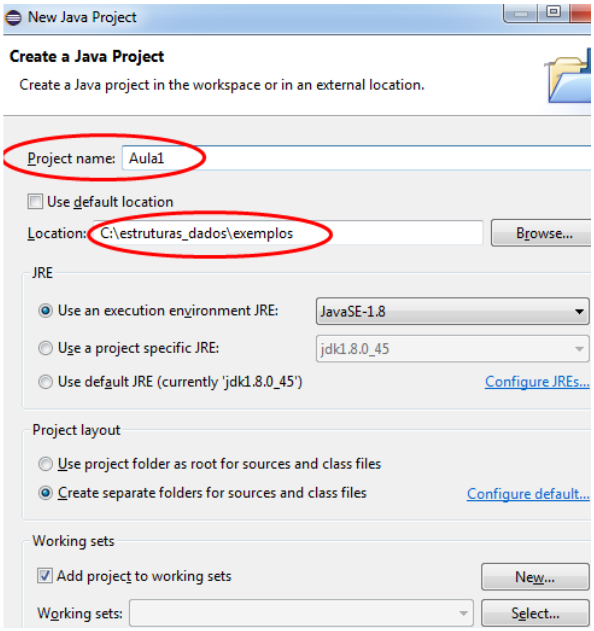

Figura 2 – Nome e local onde o projeto será salvo.

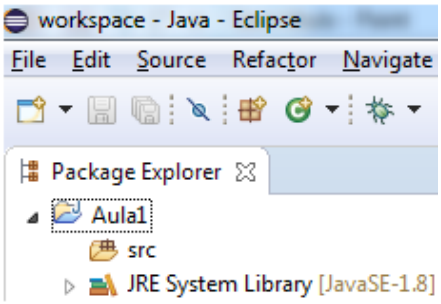

Figura 3 – Estrutura do projeto no Eclipse.

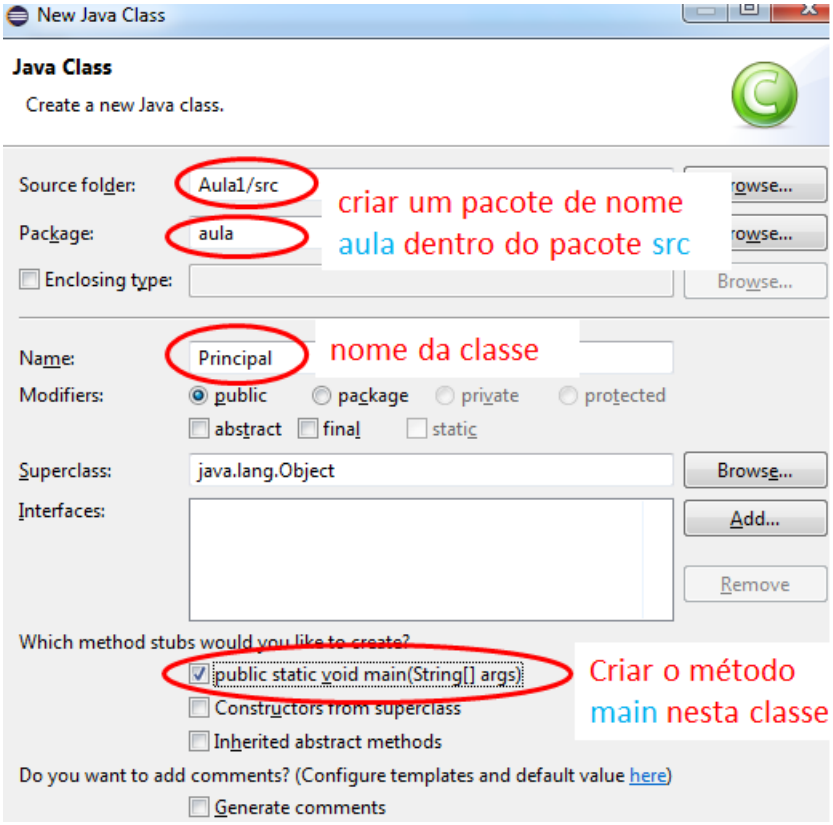

Figura 4 – Adicionar a classe Principal no projeto.

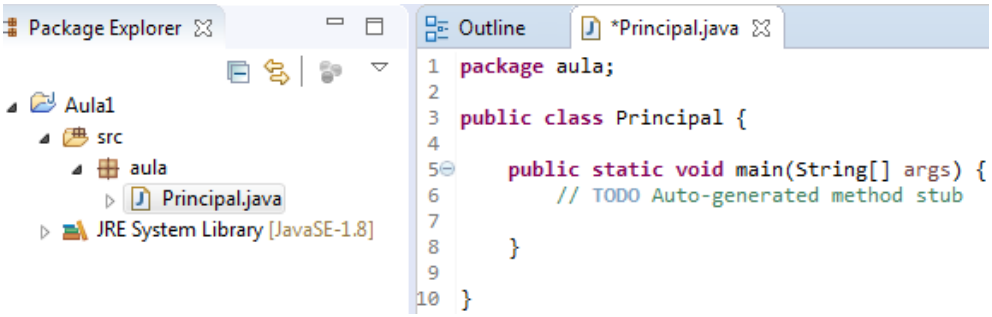

Figura 5 – Estrutura do projeto no Eclipse e a classe Principal.

2 - Exemplos

Exemplo 1 – Criar uma classe de nome Ponto no pacote aula do Projeto Aula1 assim como mostrado na Figura 7. O método imprimir deverá imprimir na tela o valor dos atributos x e y.

Passo 1 – Clique com o botão direito do mouse sobre o pacote aula e selecione New -> Class. Na janela seguinte forneça o nome da classe Ponto e finalize;

Passo 2 – Programe o corpo do método imprimir com o código da Figura 6;

Passo 3 – Programe o corpo do método main com o código da Figura 8 e na sequência execute o programa para ver o resultado.

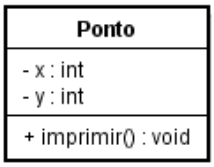

Figura 7 – Diagrama UML da classe Ponto.

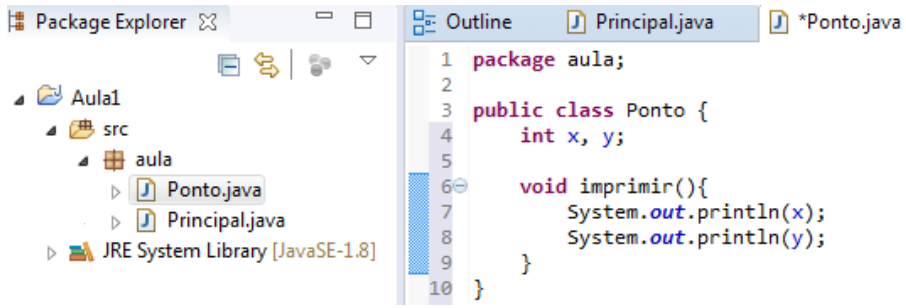

Figura 6 – Código da classe Ponto.

```java
package aula;

public class Principal {
   public static void main(String[] args) {
      /* declaração das variáveis a e b do tipo Ponto */
      Ponto a, b;
      /* criar os objetos e colocar os endereços (referências) nas variáveis a e b */
      a = new Ponto();
      b = new Ponto();
      /* os atributos x e y são inicializados com o valor padrão para números, que é zero */
      a.imprimir();
      b.imprimir();
      /* colocar valores nos atributos */
      a.x = 5;
      a.y = 3;
      b.x = 9;
      b.y = 6;
      /* o valor dos atributos será (5,3) e (9,6) */
      a.imprimir();
      b.imprimir();
   }
}
```

Figura 8 – Código da classe Principal.

Exemplo 2 – Programar os métodos distancia() e distancia(Ponto) na classe Ponto assim como mostrado no diagrama da Figura 9.

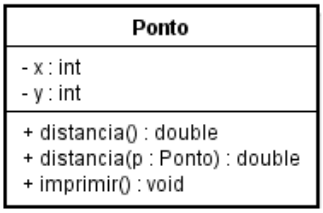

Figura 9 – Diagrama UML da classe Ponto.

A Figura 10 possui a codificação dos métodos e a Figura 11 mostra o seu uso.

A assinatura de um método é formada pelo nome do método e os tipos de dados dos seus parâmetros:

- O método distancia() possui a assinatura distancia no arguments;
- O método distancia(Ponto p) possui a assinatura distancia Ponto.

No método main (Figura 11) a chamada a.distancia() está chamando um método com a assinatura distancia no arguments, já a chamada a.distancia(b) está chamando um método com a assinatura distancia Ponto, pois b é do tipo Ponto.

Quando dois ou mais métodos possuem o mesmo nome dizemos que existe sobrecarga (overload).

```java
package aula;

public class Ponto {
   int x, y;

   /* assinatura: distancia no arguments */
   double distancia(){
      /* o método estático pow retorna x^2 */
      double dx = Math.pow(x,2);
      double dy = Math.pow(y,2);
      /* o método estático sqrt retorna a raiz quadrada */
      return Math.sqrt(dx + dy);
```

```
    }

    /* assinatura: distancia Ponto */
    double distancia(Ponto p){
        /* cálculo entre o atributo x deste objeto e o atributo x de p */
        double dx = Math.pow(x - p.x, 2);
        double dy = Math.pow(y - p.y, 2);
        return Math.sqrt(dx + dy);
    }

    void imprimir(){
        System.out.println("(" + x +", "+ y +")");
    }
}
```

Figura 10 – Código da classe Ponto.

```
package aula;

public class Principal {
    public static void main(String[] args) {
        /* declaração das variáveis a e b do tipo Ponto */
        Ponto a, b;
        /* criar os objetos e colocar os endereços (referências) nas variáveis a e b */
        a = new Ponto();
        b = new Ponto();
        /* colocar valores nos atributos */
        a.x = 5;
        a.y = 3;
        b.x = 9;
        b.y = 6;
        /* distância entre o ponto e (0,0) */
        System.out.println( a.distancia() );
        System.out.println( b.distancia() );
        /* distância entre a e b */
        System.out.println( a.distancia(b) );
        System.out.println( b.distancia(a) );
    }
}
```

Figura 11 – Código da classe Principal.

Exemplo 3 – Programar o construtor Ponto(int, int) na classe Ponto assim como mostrado no diagrama da Figura 12.

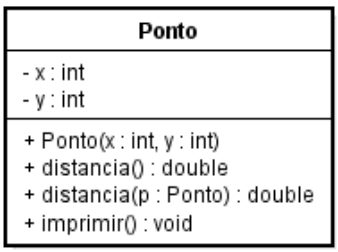

Figura 12 – Diagrama UML da classe Ponto.

A instrução this refere-se ao próprio objeto. No construtor da Figura 13, a instrução

this.x = x;

this.x refere-se ao atributo x do próprio objeto, já x é a variável que recebe o parâmetro.

No método main da Figura 14 a instrução

a = new Ponto();

causaria erro, pois não existe um construtor com a assinatura Ponto no arguments na classe Ponto.

```
package aula;

public class Ponto {
    int x, y;

    /* construtor da classe */
    Ponto(int x, int y){
        /* o construtor é usado para inicializar os atributos */
        this.x = x;
        this.y = y;
    }
```

```
    double distancia(){
       double dx = Math.pow(x,2);
       double dy = Math.pow(y,2);
       return Math.sqrt(dx + dy);
    }

    double distancia(Ponto p){
       double dx = Math.pow(x - p.x, 2);
       double dy = Math.pow(y - p.y, 2);
       return Math.sqrt(dx + dy);
    }

    void imprimir(){
       System.out.println("(" + x +", "+ y +")");
    }
}
```

Figura 13 – Código da classe Ponto.

```
package aula;

public class Principal {
   public static void main(String[] args) {
      Ponto a, b;
      /* é necessário fornecer dois inteiros no construtor */
      a = new Ponto(5, 3);
      b = new Ponto(9, 6);
      /* distância entre o ponto e (0,0) */
      System.out.println( a.distancia() );
      System.out.println( b.distancia() );
      /* distância entre a e b */
      System.out.println( a.distancia(b) );
      System.out.println( b.distancia(a) );
   }
}
```

Figura 14 – Código da classe Principal.

Exemplo 4 – Programar um array de inteiros e outro do tipo Ponto na classe Principal.

A Figura 15 mostra as intruções para criar, preencher e imprimir os arrays na tela. Veja que a codificação possui sintaxes semelhantes para tipos de dados primitivos e objetos, a diferença está apenas no tipo de conteúdo de cada elemento do array.

```
package aula;

public class Principal {
   public static void main(String[] args) {
      /* declaração de uma variável para receber endereços de array de int*/
      int[] v;
      /* criação de um array de int com 4 elementos */
      v = new int[4];
      /* preencher o array - é necessário acessar uma posição por vez */
      v[0] = 11; /* a 1a posição possui índice zero */
      v[1] = 14;
      v[2] = 17;
      v[3] = 20; /* a última posição possui índice n-1 */
      /* imprimir o array - é necessário acessar uma posição por vez */
      for( int i = 0; i < v.length; i++ ){
         System.out.println( v[i] );
      }

      /* declaração de uma variável para receber endereços de array de Ponto*/
      Ponto[] w;
      /* criação de um array de Ponto com 4 elementos */
      w = new Ponto[4];
      /* preencher o array */
```

```
        w[0] = new Ponto(2,3); /* é necessário criar um objeto para cada posição do array */
        w[1] = new Ponto(4,1);
        w[2] = new Ponto(5,2);
        w[3] = new Ponto(4,5);
        /* imprimir o array - é necessário acessar uma posição por vez */
        for( int i = 0; i < w.length; i++ ){
           /* cada posição do array possui um objeto do tipo Ponto, logo, existe um método imprimir em w[i] */
           w[i].imprimir();
        }
    }
}
```

Figura 15 – Código da classe Principal.