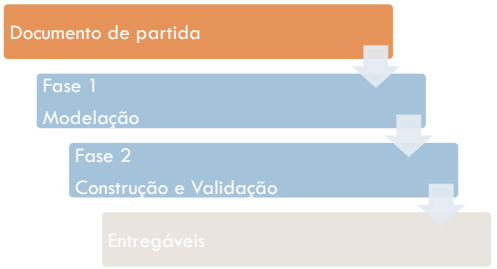
PROGRAMAÇÃO ORIENTADA PELOS OBJECTOS

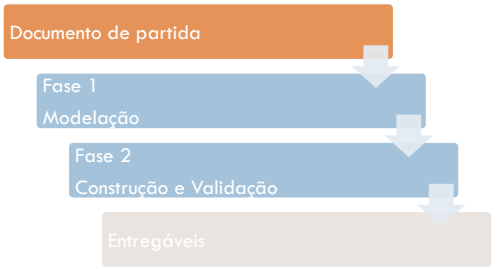
Implementação de interfaces e documentação

Implemente as suas classes



- Escolha uma ordem que facilite o desenvolvimento incremental da aplicação
- Evite deixar "pontas soltas"
- Lembre-se **sempre** da regra da versão estável
- Escolha nomes informativos (vars, classes, métodos, etc)
- Vá comentando o seu código à medida que o escreve
  - A experiência mostra que comentários deixados para o fim acabam por não ser feitos, ou são feitos à pressa
  - É saudável comentar o cabeçalho dos métodos antes de os desenvolver. Leva-nos a pensar no que temos de fazer, antes de começar a programar

Implemente as suas classes



- Escolha uma ordem que facilite o desenvolvimento incremental da aplicação
- Evite deixar "pontas soltas"
- Lembre-se **sempre** da regra da versão estável
- Escolha nomes informativos (vars, classes, métodos, etc)
  - How to name things: the hardest problem in programming
- Vá comentando o seu código à medida que o escreve

Convenções de documentação

- Escreva a documentação a pensar nas pessoas que vão ter de usar o seu código e mantê-lo
  - Documentar **interface pública** para que outros o possam **usar** de forma correcta e eficiente
  - Documente **interface privada** e os detalhes de implementação internos, para que outros possam **manter e melhorar** o seu código

Convenções de documentação

- Assuma sempre que alguém completamente estranho ao seu código o vai ter de entender, mais cedo ou mais tarde!
  - Se passar tempo suficiente, até o autor de código mal documentado terá dificuldade em o compreender
- Mantenha o seu código e comentários sincronizados
  - Ambos fazem parte do produto, não menospreze nenhum dos dois
- Seja claro, conciso e preciso

Três tipos de comentário, em Java

- Comentário de documentação
  - Começa em /** e termina em */
  - É o conteúdo destes comentários que alimenta a documentação Javadoc
- Comentário standard (estilo emprestado do C)
  - Começa em /* e termina em */
- Comentário de uma linha
  - Começa em // e termina no fim da linha

Comentário de documentação

- Pode ser usado para comentar qualquer elemento
  - Declaração de interface, classe, método, construtor, campo
- Estes comentários dão a informação necessária para que o Javadoc possa gerar documentação em formato html (ao estilo que costuma ver na documentação da API do Java)

Comentário de documentação

- O Javadoc apenas reconhece estes comentários quando estes aparecem imediatamente antes do elemento a comentar.
  - Não use este formato para comentar detalhes de implementação
- O Javadoc apenas reconhece um bloco de comentários por elemento
- O principal objectivo destes comentários é definir um contrato entre o cliente e o fornecedor de um serviço

Comentários standard (estilo C)

- Use este estilo de comentário **apenas para esconder temporariamente do compilador código**
  - Relembrando: começa em /* e termina em */

Comentário de uma linha `//line comm`

- Use este tipo de comentário para explicar detalhes de implementação
  - Destina-se à comunicação entre programadores
  - Use um ou mais comentários destes para documentar
    - O objectivo de variáveis ou expressões
    - Decisões ao nível da implementação
    - Informação adicional sobre algoritmos complicados
    - "Remendos" aplicados a defeitos
    - Código a necessitar de melhorias/optimização
    - Problemas conhecidos, limitações, ou deficiências

Documentação

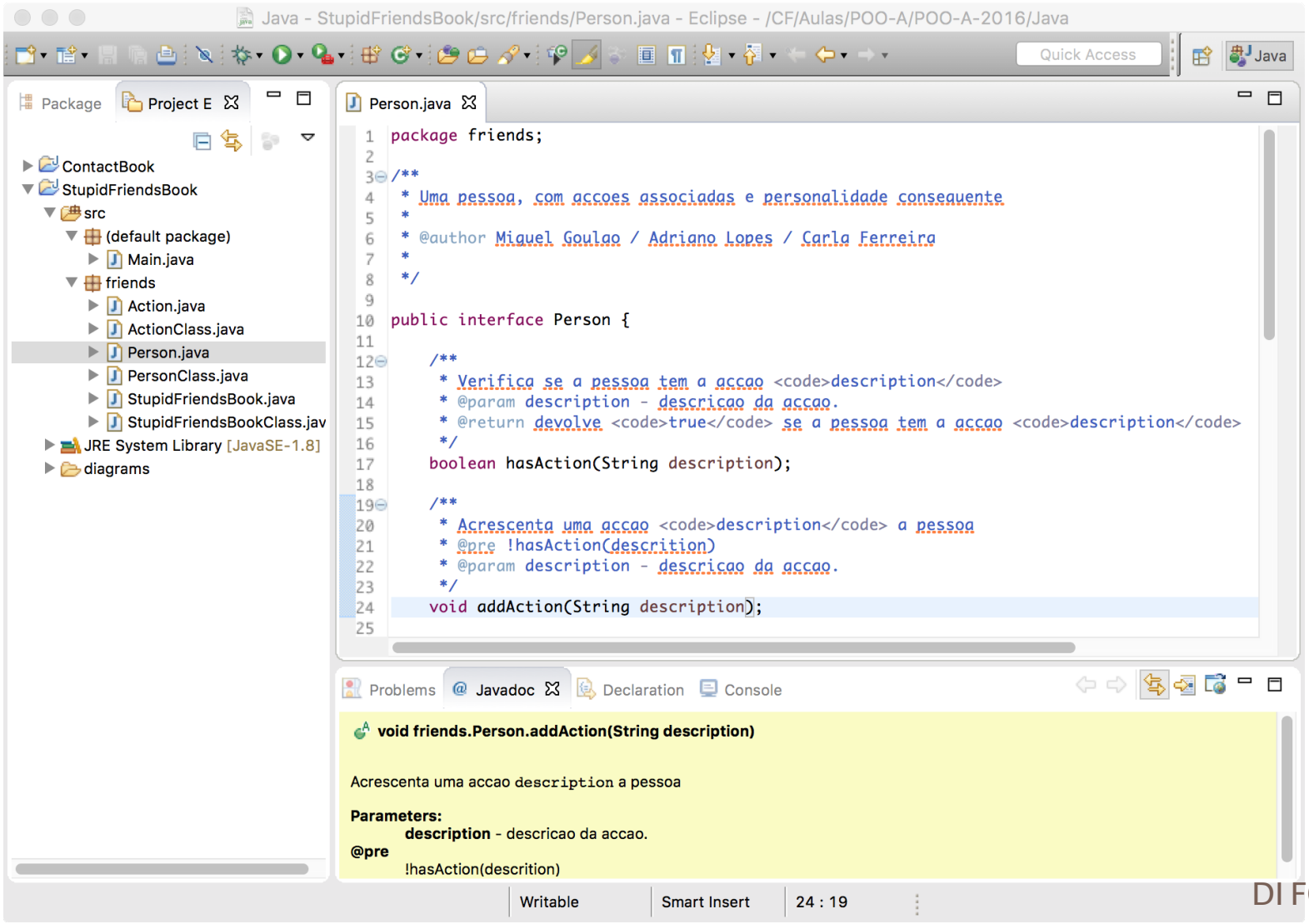
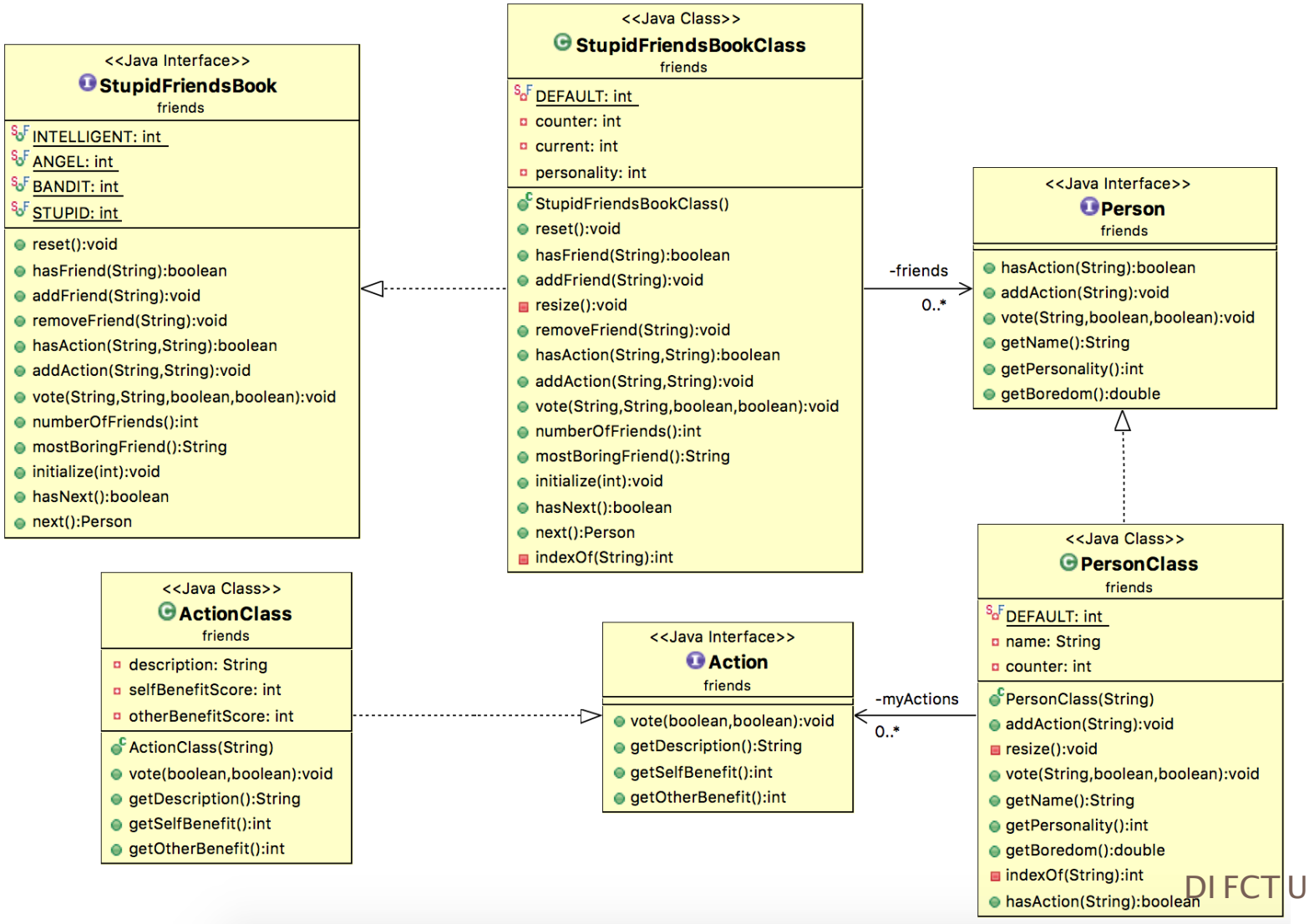
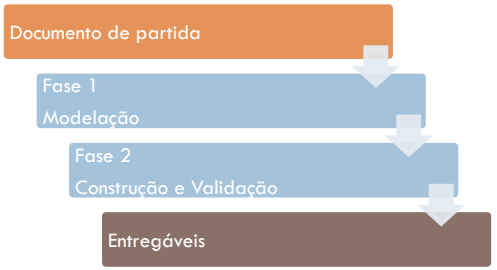


Diagrama de classes e interfaces



Entregáveis



- Código fonte do projecto
  - (default package)
    - Main.java
  - Package `poo;`
    - StupidFriendsBook.java
    - StupidFriendsBookClass.java
    - Person.java
    - PersonClass.java
    - Action.java
    - ActionClass.java
- Documentação em formato Javadoc
- Outra documentação (e.g. Diagrama de classes)

Facturas

Requisitos

- Escreva um programa que permita imprimir uma factura, com base em dados lidos de um ficheiro
  - Uma factura descreve os valores cobrados por determinados produtos, cada um dos quais com determinadas quantidades
  - Uma factura é composta por linhas
  - Cada linha tem informação sobre um produto, incluindo a sua descrição, preço unitário, quantidade e preço total
  - A factura apresenta um endereço no início e o total a pagar, no fim
- Neste exemplo, vamos ignorar impostos, números fiscais e outras complicações

Exemplo de ficheiro de dados


```
Pastelaria Doces com Bué Açúcar
3
Pastel de Nata
1.2 3
Croissant com chocolate
1.5 4
Sorbet de limão em prosecco
3.2 2
```

Exemplo de factura


```
Pastelaria Doces com Bué Açúcar
descrição; preço unitário; quantidade; total
Pastel de Nata; 1.20; 3; 3.60
Croissant com chocolate; 1.50; 4; 6.00
Sorbet de limão em prosecco; 3.20; 2; 6.40
16.0
```

Entrada e saída de dados

- Ler dados de consumo
- Imprimir factura

Entidades

- Factura
  - Adicionar linhas à factura
  - Escrever factura
    - Obter endereço
    - Listar linhas detalhadas
    - Devolver total a pagar
- Linha de factura
  - Devolver produto, quantidade e total
- Produto
  - Devolver descrição do produto e preço unitário

Como especificar estas entidades?

- Directamente no código fonte?
  - Especificação das interfaces e das classes que as implementam
  - Comentários no código
    - Usando o Javadoc
- Através de uma notação diagramática
  - Uma das mais populares é o UML, um *standard* usado na indústria para a concepção (*design*) de *software* orientado a objectos
    - Nesta disciplina, usaremos diagramas de classes, que são apenas um dos mais de 10 tipos de diagramas do UML

Invoice (Interface documentada)

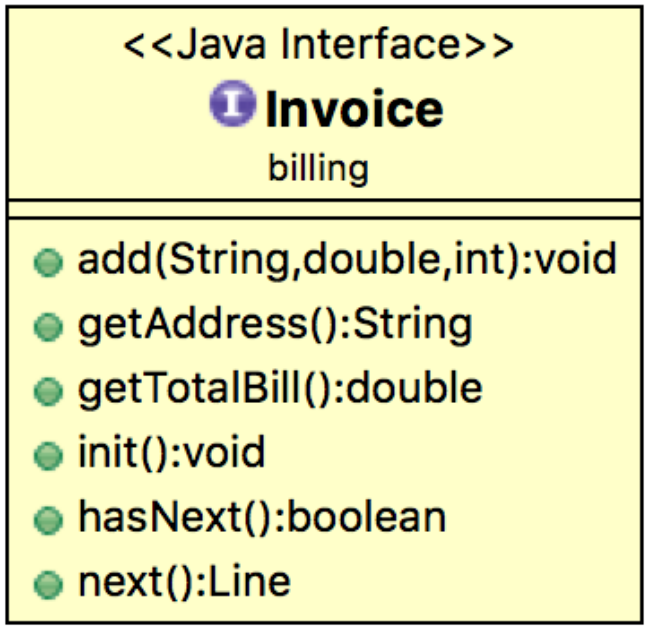

```
package billing;
public interface Invoice {
  /**
   * Adiciona uma linha à factura, com <code>n</code> produtos com
   * descrição <code>description</code> e preço <code>price</code>.
   * @param description descrição do produto
   * @param price preço unitário do produto
   * @param n quantidade de items adicionados nesta linha.
   */
   void add(String description, double price, int n);
  /**
   * Devolve o endereço do emissor da factura
   * @return o endereço do emissor da factura
   */
   String getAddress();
   /**
    * Calcula o total da factura
    * @return o total da factura
    */
   double getTotalBill();
   // Continua...
```

Invoice (Interface documentada)


```
package billing;
public interface Invoice {
    // ... continuação
    /**
     * Inicializa o iterador de linhas da factura
     */
    void init();

    /**
     * Verifica se existe mais alguma linha na factura
     * @return <code>true</code>, se existe,
     * <code>false</code> caso contrário
     */
    boolean hasNext();
    /**
     * Devolve a próxima linha na factura
     * @pre hasNext()
     * @return a próxima linha da factura
     */
    Line next();
}
```

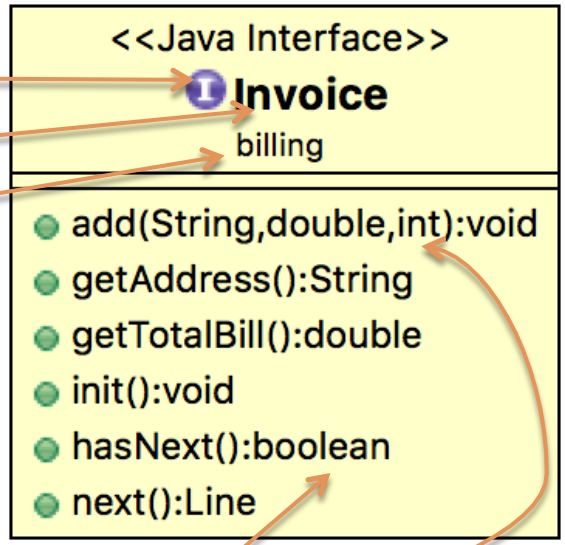
Invoice (Interface em Java e em UML)


```
package billing;
public interface Invoice {
    // Ignorando os comentários javadoc
    void add(String description, double price, int n);
    String getAddress();
    double getTotalBill();
    void init();
    boolean hasNext();
    Line next();
}
```



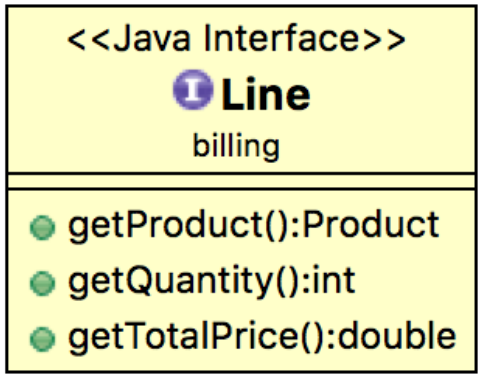
Especificação diagramática de uma interface

- Icon simbolizando a interface
- Nome da interface
- Nome do pacote
- Operações definidas na interface
- Lista de parâmetros
- Tipo de retorno das operações

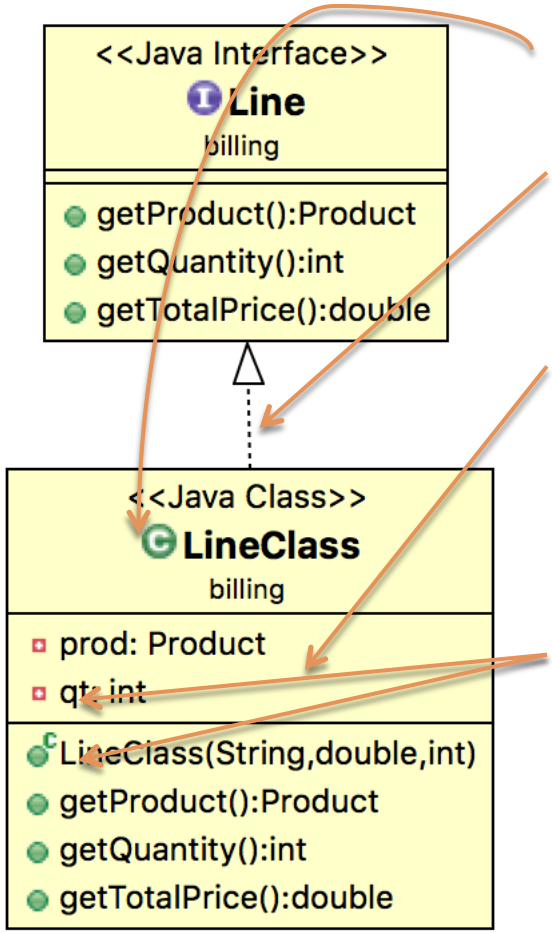


Line (Interface em Java e em UML)


```
package billing;
public interface Line {
   /**
    * Devolve o produto associado à linha
    * @return o produto da linha
    */
   Product getProduct();
   /**
    * Devolve a quantidade de produtos
    * @return número de unidades nessa linha
    */
   int getQuantity();
   /**
    * Devolve o preço total dos produtos nesta linha
    * @return o preço total dos produtos
    */
   double getTotalPrice();
}
```



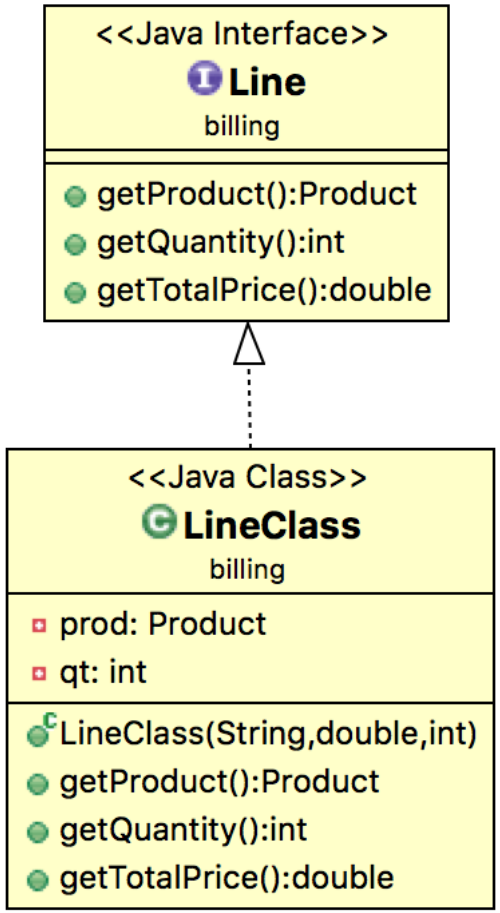
Classes que implementam interfaces



- Classe decorada com um icon diferente
- Relação implements representada com uma seta fechada e linha tracejada
- Repare no compartimento extra, para a declaração de variáveis e constantes na classe
- Icons diferenciados para mostrar visibilidade privada e pública, na classe

Classes que implementam interfaces

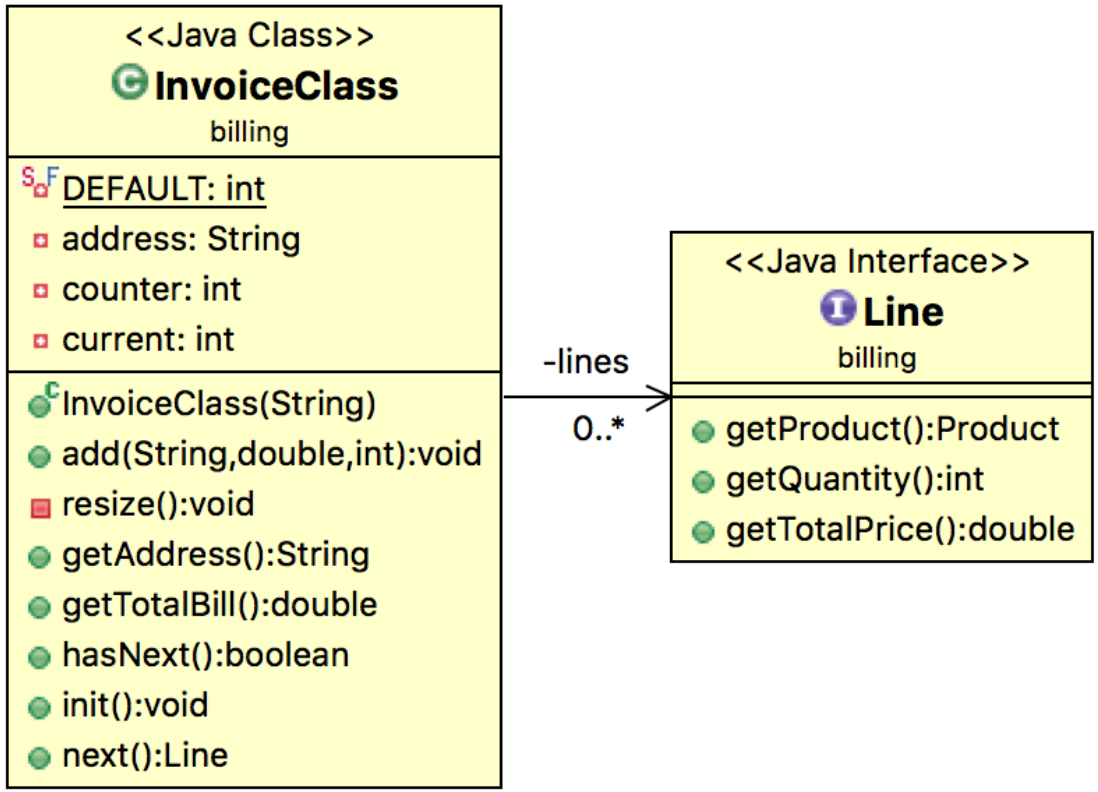

```
package billing;

class LineClass implements Line {
  private Product p;
  private int n;

  public LineClass(String d,
                   double p, int n) {...}
  public Product getProduct() {...}
  public int getQuantity() {...}
  public double getTotalPrice() {...}
}
```



Agregação

- Usa-se quando um elemento contém uma **colecção** de outros elementos de um determinado tipo

- Nome da variável, do lado do elemento "contido"



Dependência

- Usado quando uma interface, ou classe, depende de outra para o seu comportamento
  - Main usa o tipo Invoice

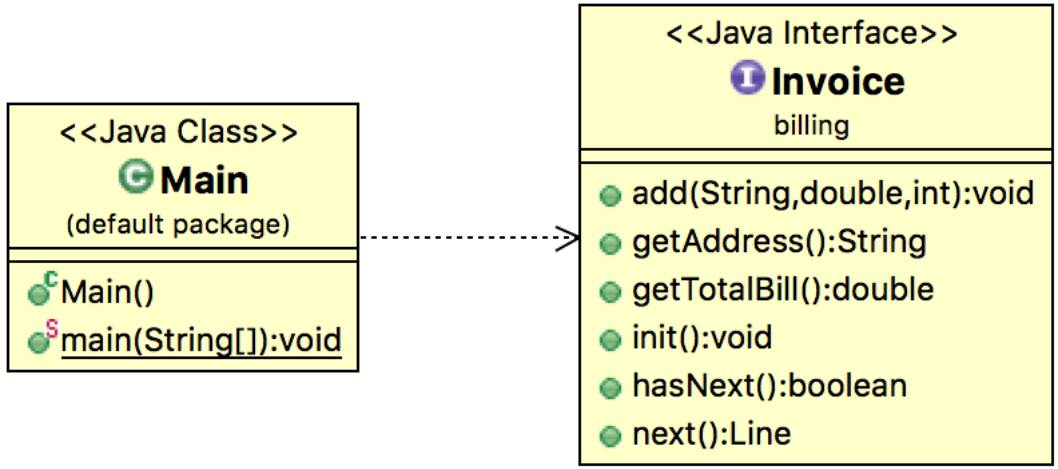
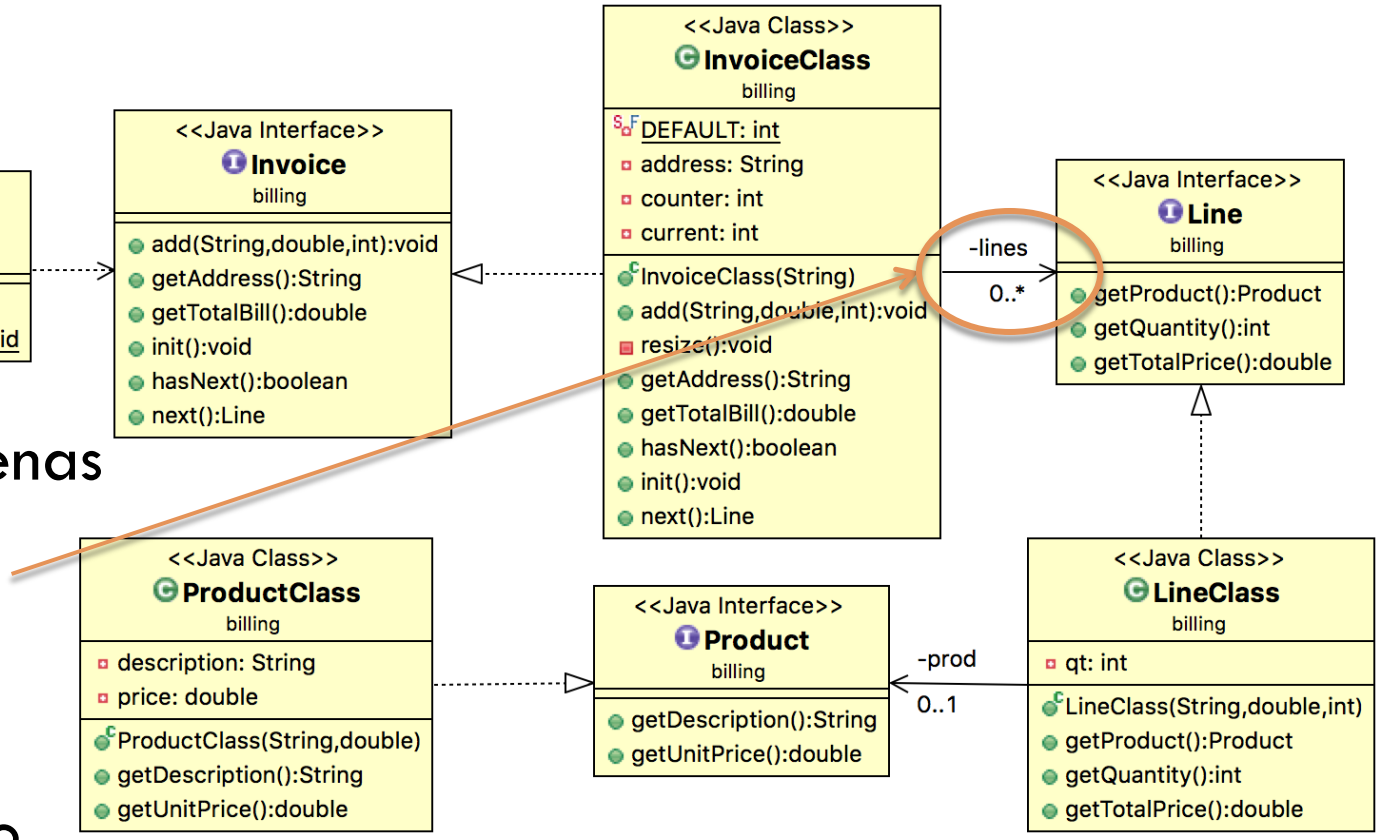


Diagrama de classes e interfaces



- Notem que apenas classes podem originar agregações. Interfaces não podem pois não possuem estado

Demo do plug-in para o ObjectAid UML Diagram

Este plug-in, instalado nos laboratórios, permite gerar os diagramas a partir do código e vice-versa, dentro do próprio projecto do Eclipse.