***Importante***: Para cada item abaixo deve ser copiado trechos do código que cumprem o requisito e explicado, se não for aparente, o porquê o requisito é cumprido. Sejam bem explícitos. Deve ser indicado também o arquivo da classe em que está o trecho do código. Eu avaliarei o código do Github a partir desse documento para confirmá-lo e também para detectar possíveis erros. **Quem não seguir o que está indicado aqui, não terá o projeto avaliado e perderá a atividade.**

**Usar um novo repositório!**

**Entrega final: 26 de novembro via GitHub. Prova oral sobre o código ainda a ser definida.**

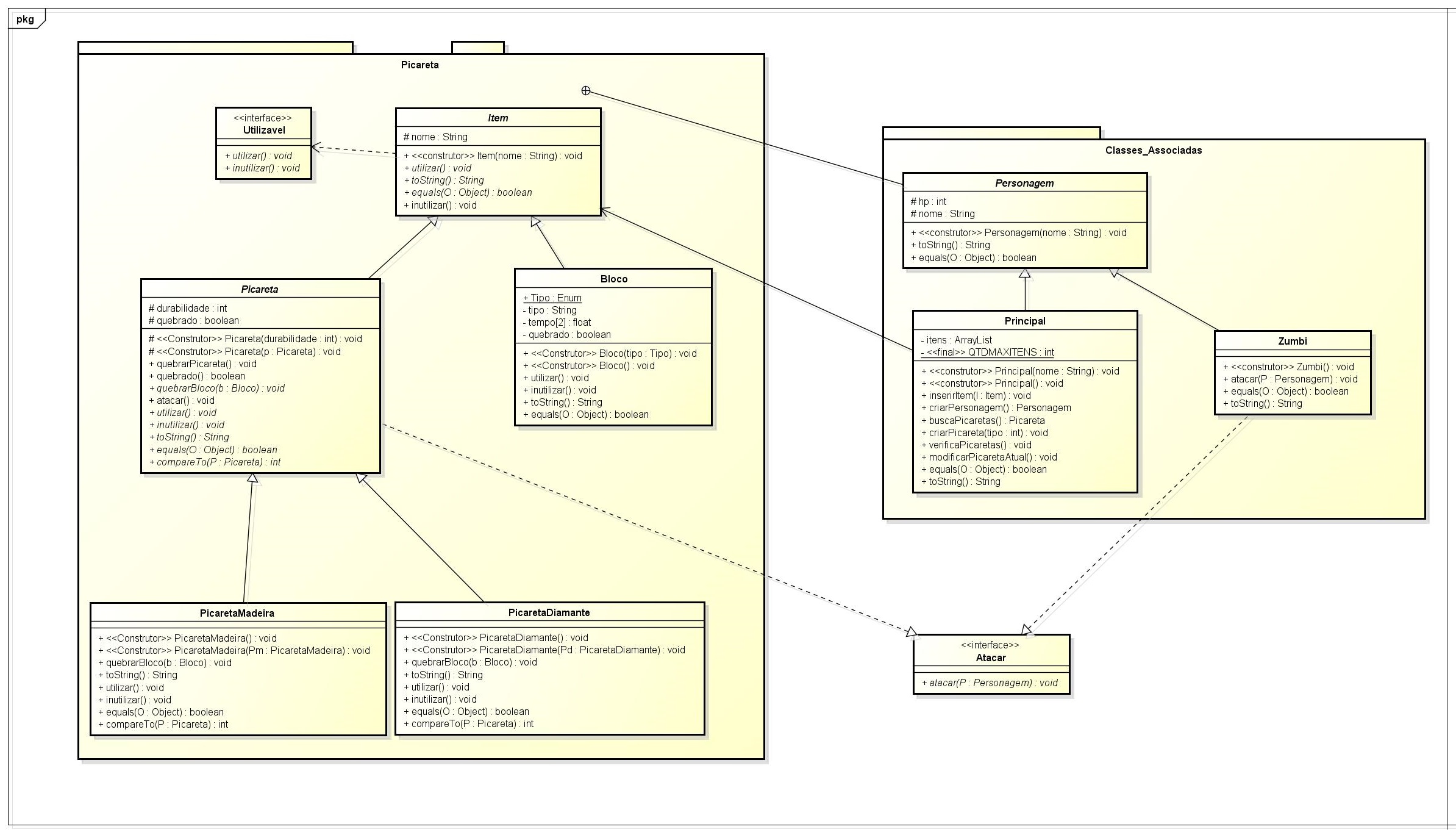
**Entrega parcial: 19 de novembro via GitHub. Apenas Diagrama de Classes, interfaces e classes abstratas no branch general.**

**Requisitos Gerais**

***Todos os atributos e funções membros devem estar relacionados a classe***

1. Pelo menos 4 atributos
2. Pelo menos 4 funções membros sem incluir get e set
3. Diagrama UML completo (obrigatório salvar também o png do diagrama no gitHub)

- Como pode ser visto no diagrama abaixo, todos os requisitos acima sao cumpridos:

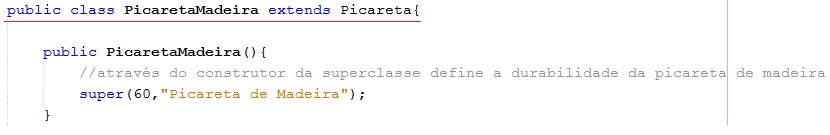


**Requisitos de implementação**

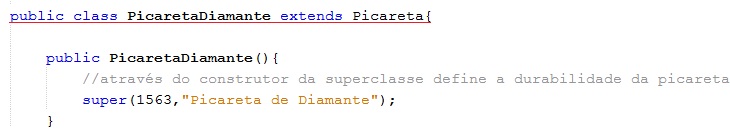
1. Todas as classes concretas devem vir de interfaces ou classes abstratas. Pelo menos três hierarquias de classes. Uma das hierarquias deve ter três níveis. Exemplo: Personagem >> Ciborgue >> Robocop; Class Arma (interface) >> Beretta93R

Como pode ser visto no diagrama de classes acima, as classes concretas “PicaretaMadeira”, “PicaretaDiamante”, “Bloco”, herdam da classe abstrata “Item”, e as classes “Principal” e “Zumbi” herdam da classe abstrata “Personagem”.

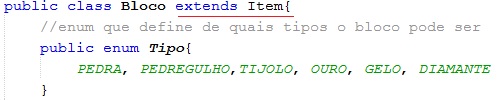
PicaretaMadeira.java:



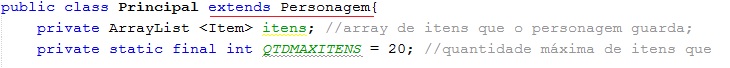
PicaretaDiamante.java:



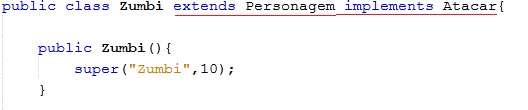
Bloco.java:



Principal.java:



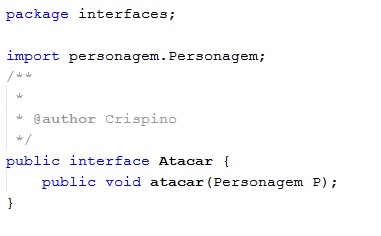
Zumbi.java:



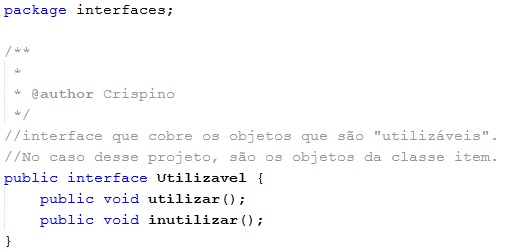
1. Ao menos três interfaces. A terceira interface deve ser uma interface que liga duas hierarquias como no exemplo da interface **corredor** (Figura 1).

- Requisito parcialmente cumprido. Foram criadas duas interfaces: “Atacar” e “Utilizável”.

Atacar.java:

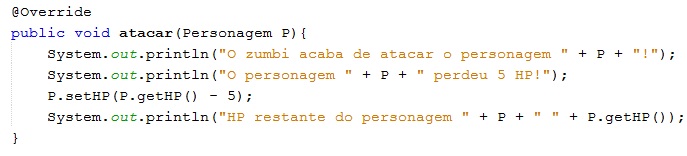
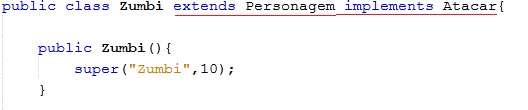


Utilizavel.java:

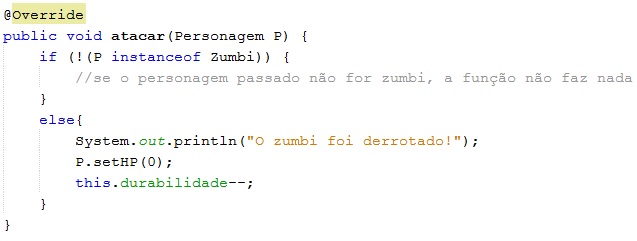


Como pode ser visto nas imagens abaixo, a interface “Atacar” liga a duas hierarquias diferentes: “Picareta” implementa “Atacar”, assim como “Zumbi” a implementa:

Zumbi.java:



Picareta.java:



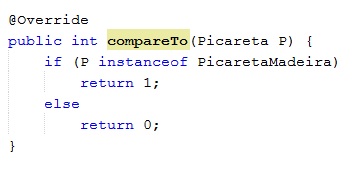
1. Usar a interface **Comparable** e sobrescrever o método **compareTo** em pelo menos uma hierarquia

- A interface Comparable é implementada na classe “Picareta”.

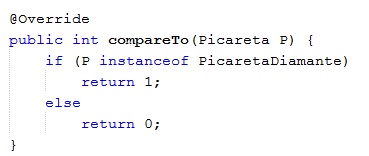
Picareta.java:



PicaretaMadeira.java:



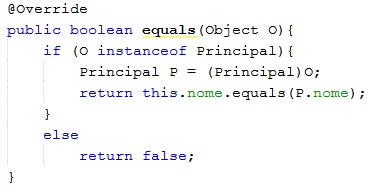
PicaretaDiamante.java:



1. Sobrescrever **equals** para de Object

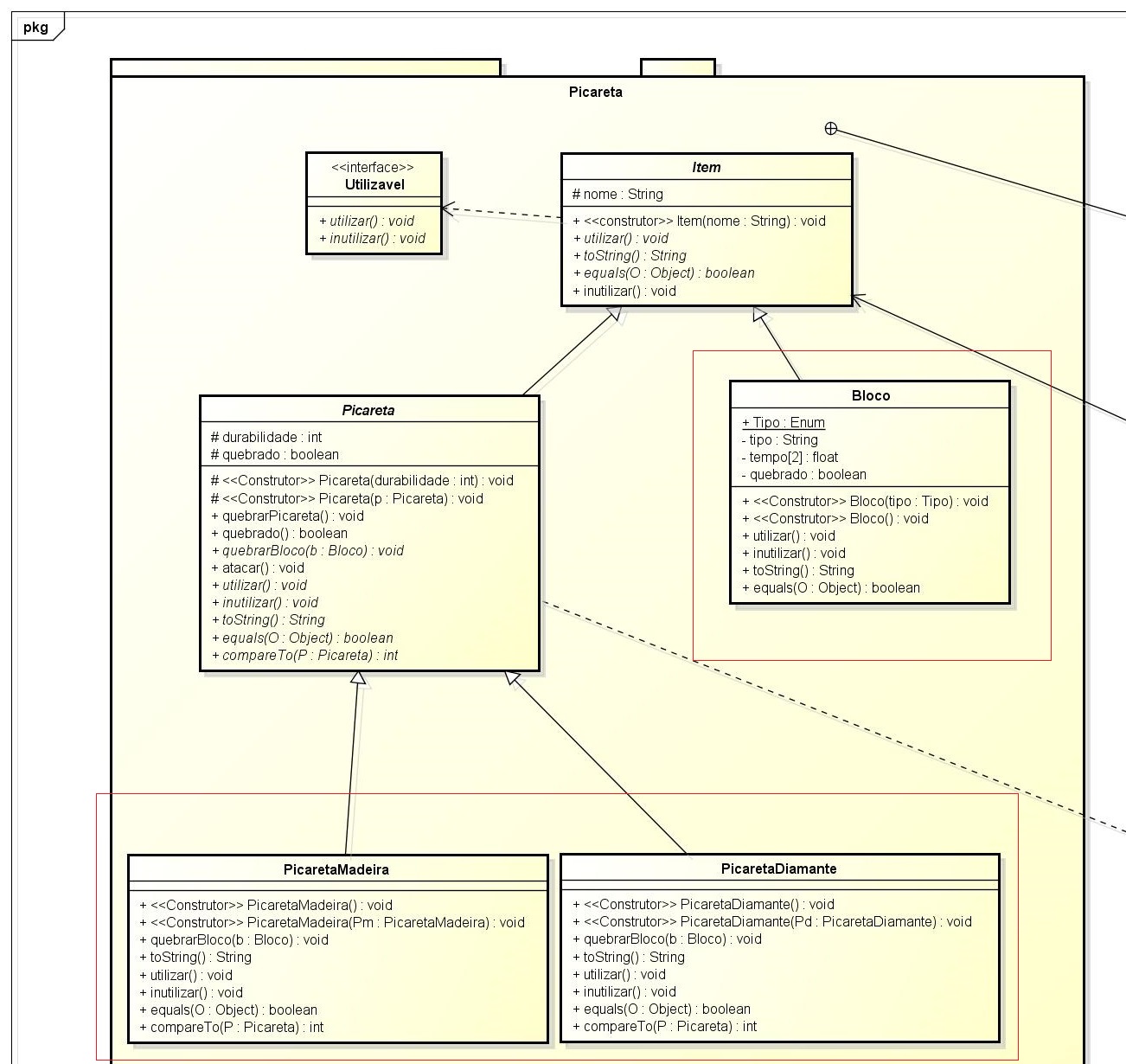
O método equals é sobrescrito em todas as classes. Segue abaixo um exemplo da classe “Principal”:

-Principal.java:



1. Todas as hierarquias devem ter classes Concretas, e em uma das hierarquias, três classes Concretas relacionadas: Exemplo Servico >> ServicoStream >> (Netflix, HBOStream, AmazonPrime, NowTv). Em uma pesquisa de 10 segundos: <http://www.tomsguide.com/us/pictures-story/620-top-online-streaming-video.html>

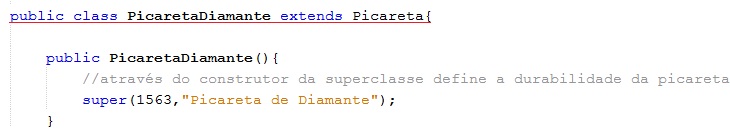
- Na hierarquia da classe “Item”, 3 classes concretas estao relacionadas: “PicaretaMadeira”, “PicaretaDiamante” e “Bloco”:



1. Sempre usar o super para o máximo de reaproveitamento de código

- “super” utilizado em todos os construtores de subclasses. Como exemplo, segue abaixo o construtor da classe “PicaretaDiamante”:

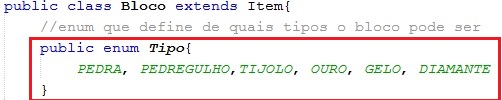
PicaretaDiamante.java:



1. Atributos static e const static

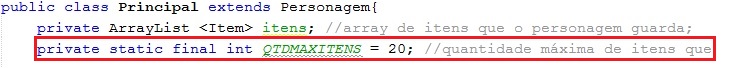
- Static: variável enum de nome “Tipo” na classe Bloco.

Bloco.java:



- Const static: variável “*QTDMAXITENS*”, localizada na classe “Principal”.

Principal.java:

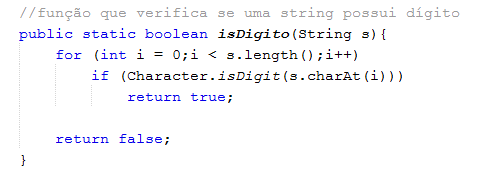


1. Método static

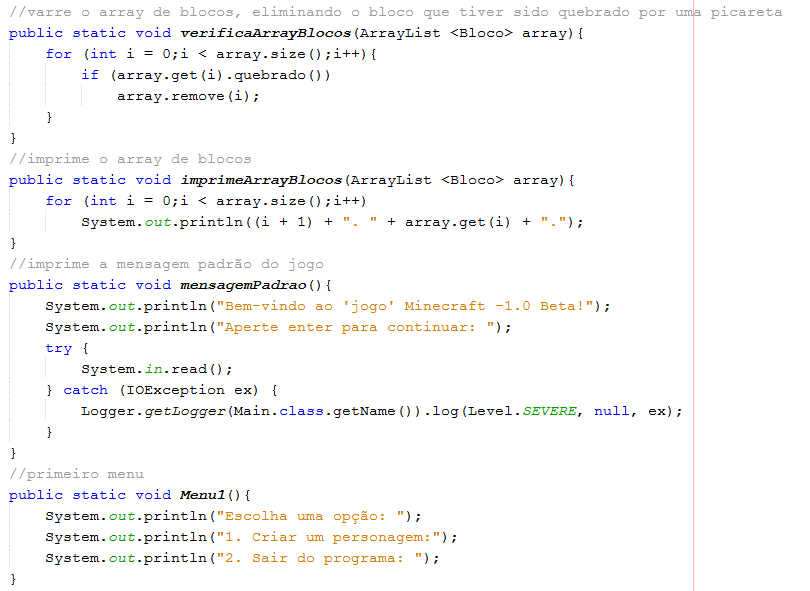
- Diversos métodos statics são implementados e utilizados no main(Main.java). Seguem alguns como exemplo:

Main.java:

*isDigito()*:



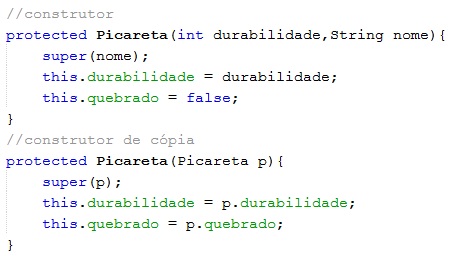
*verificaArrayBlocos()*, *imprimeArrayBlocos()*, *mensagemPadrao()* e *Menu1()*:



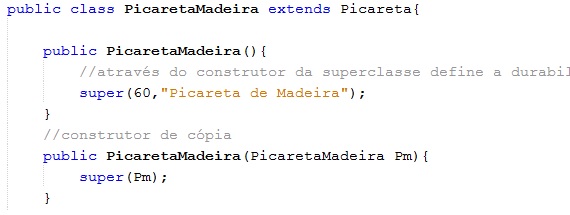
1. Construtores em todas as classes, e dois na hierarquia principal. Sempre validar os dados em todas as classes

- Como pode ser visto no item 1, todas as classes tem construtor e, abaixo, são mostrados os dois construtores localizados em cada classe da hierarquia principal:

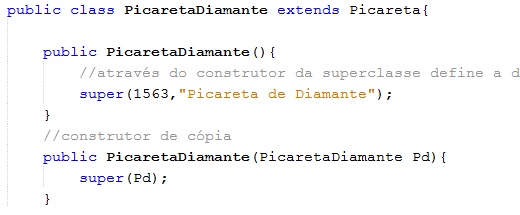
Picareta.java:



PicaretaMadeira.java:



PicaretaDiamante.java:



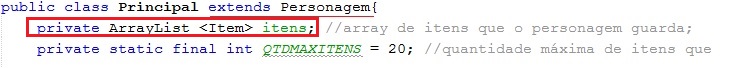
1. Construtor cópia em uma das hierarquias

- Como pode ser visto no item anterior, na classe “Picareta” e nas suas subclasses, existe um construtor de cópia.

1. ArrayList

- Na classe “Principal”, há um ArrayList de objetos da classe “Item”.

Principal.java:

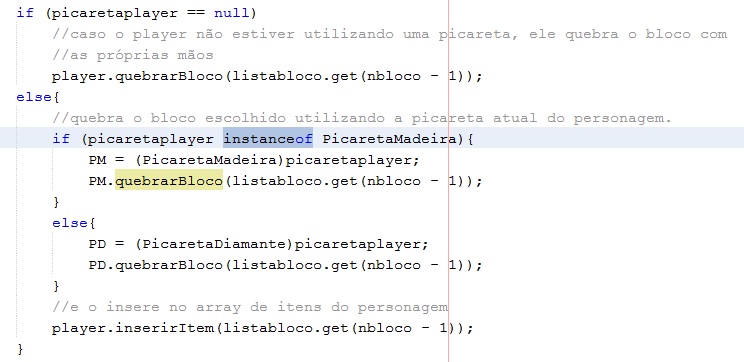


1. ENUM

- Como pode ser visto no item 7, há uma variável ENUM sendo utilizada na classe “Bloco”(Bloco.java).

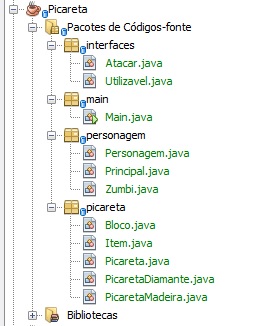
1. Usar o **instanceof** no main junto com as classes concretas. Para uma da classe concreta identificada, chamar um método dessa classe e fazer uma ação;

- No momento que o personagem principal decide quebrar um bloco com a sua picareta, o programa, através do *instanceof*, “descobre” qual é o tipo de picareta do personagem(“PicaretaMadeira” ou “PicaretaDiamante”) e chama a função “*quebrarBloco()*”, presente em ambas as classes.



1. Dividir o projeto em pacotes

- O projeto é dividido em 3 pacotes: “interfaces”, que contém as duas interfaces utilizadas no projeto; “main”, que contém a classe principal “Main.java”; “personagem”, que armazena as classes pertencentes à hierarquia da classe “Personagem”; E por último, “picareta”, que se refere às classes que estão dentro da hierarquia da classe “Picareta”.



1. Sobrescrever para todas as classes o método toString

-Segue abaixo a implementação da sobrescrita do método toString para todas as classes do meu projeto:

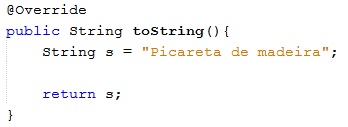
- Item.java:



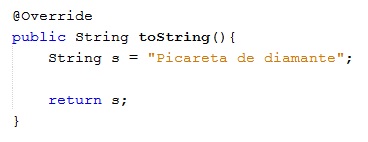
- Picareta.java:



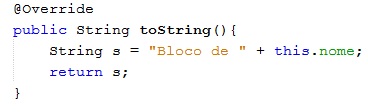
- PicaretaMadeira.java:



- PicaretaDiamante.java:



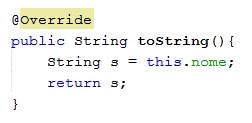
- Bloco.java:



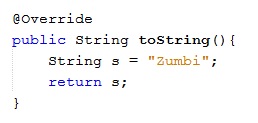
- Personagem.java:



- Principal.java:

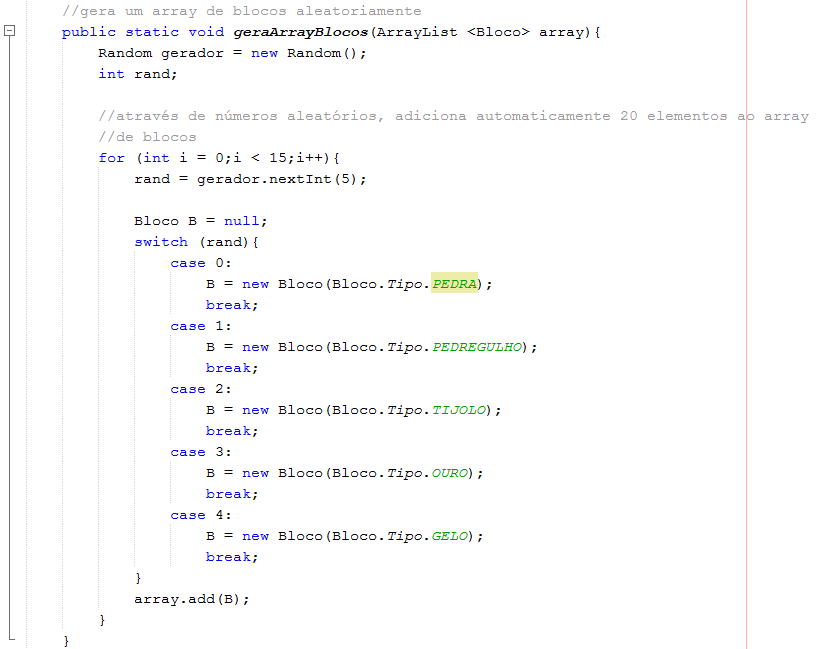


- Zumbi.java:



1. Usar a classe Random do pacote java.util (java.util.Random). Nota: deve ser usado conforme o contexto do projeto. Se for usado em um método genérico sem relação com a classe e apenas para cumpri-lo, esse requisito será desconsiderado.

- No método static “*geraArrayBlocos()*” é usado um número gerado aleatoriamente a cada iteração do laço *for* para decidir qual tipo de bloco será gerado.

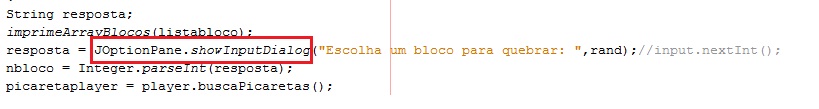


1. No main o usuário deve fazer entrada via teclado e interagir com a aplicação. Opcional de bônus: pode ser usada a classe JOptionPane do pacote javax.swing. Vejam: showInputDialog e showMessageDialog.

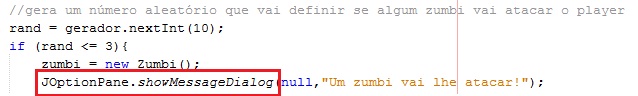
- Em diversas situações o usuário interage com o programa pelo teclado. Segue dois exemplos abaixo: o primeiro é no momento de quebrar um bloco, o usuário escolhe qual dos blocos exibidos ele quer quebrar(é usado o showInputDialog). O outro exemplo acontece quando um zumbi lhe ataca, aparece um showMessageDialog para lhe mostrar isso.

- Trecho de codigo:

- Exemplo do bloco:

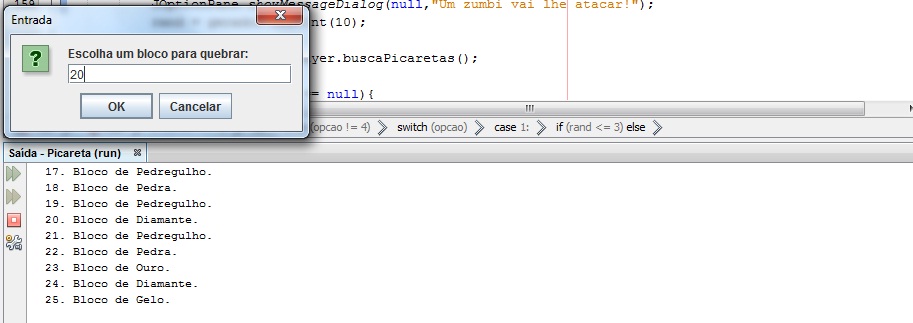


- Exemplo do zumbi:



- Execução:

- Exemplo do bloco:



- Exemplo do zumbi:

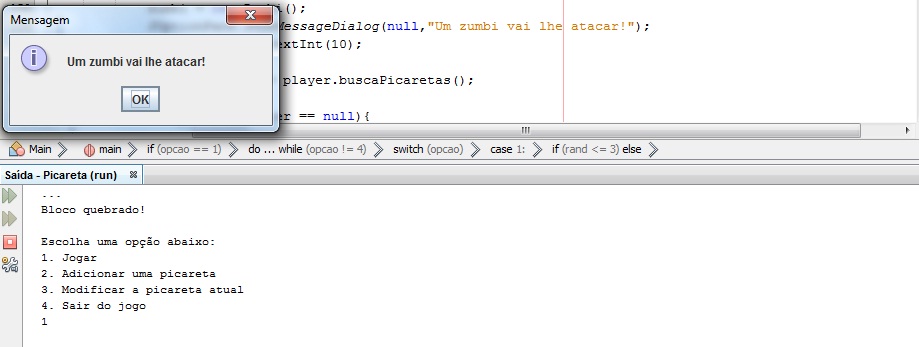


Figura 1 – Interface Corredor conectando duas hierarquias de classe