Importante: Para 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.**

Requisitos de implementação

Todos os atributos e funções membros devem estar relacionados a classe

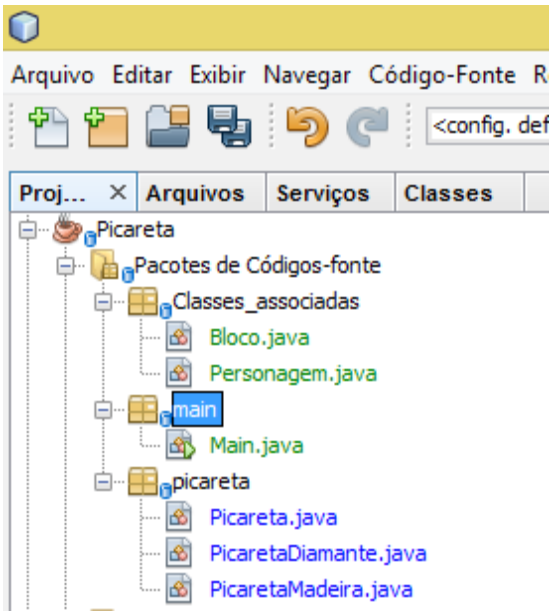
1.Pelo menos 4 atributos

2.Pelo menos 4 funções membros sem incluir get e set

Vendo os diagramas de classe no item 5, percebe-se que esses requisitos foram cumpridos.

Requisitos de implementação

3.Cinco classes: uma superclasse e duas subclasses, e duas classes relacionadas ao projeto



Como pode ser visto na imagem, há 5 classes: A superclasse Picareta, que possui 2 subclasses, PicaretaDiamante e PicaretaMadeira, e as classes associadas Bloco e Personagem, fora a classe Main.

A descrição de cada classe pode ser encontrada como comentário no começo do código-fonte de cada uma.

Essas classes estão divididas em 3 pacotes: o pacote "picareta", que contém as classes relacionadas a uma picareta do jogo Minecraft, o pacote "Classes_associadas", que guarda as outras classes associadas

que são utilizadas no projeto, e o pacote "main", que contém a classe Main.

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

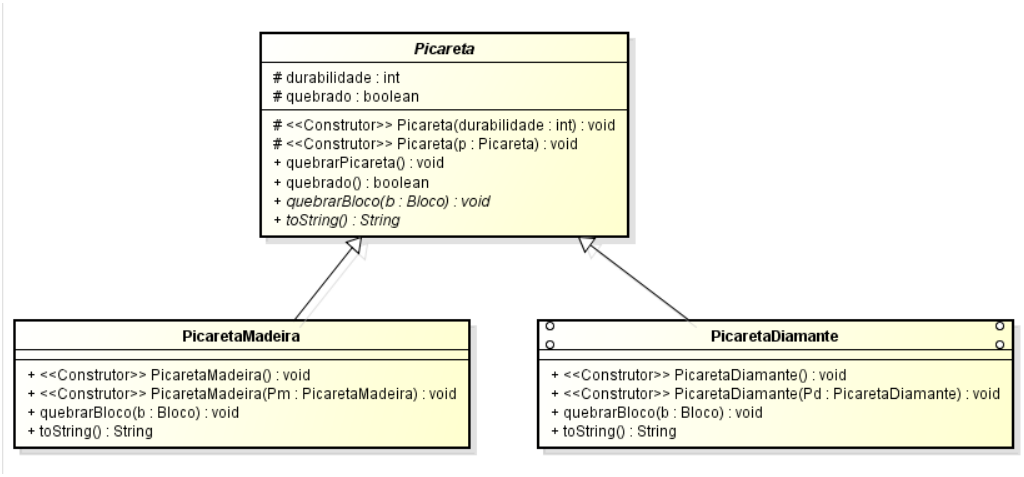
```
/**...4 linhas */
public class PicaretaDiamante extends Picareta{

    public PicaretaDiamante(){
        //através do construtor da superclasse define a durabilidade da picareta de madeira
        super(1563);
    }
```

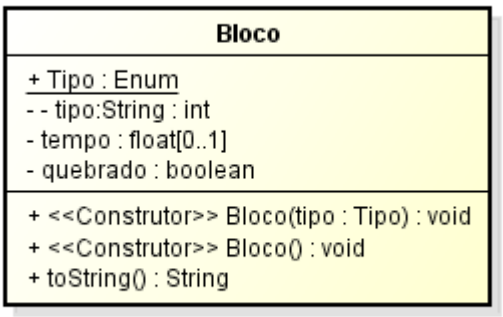
Como pode ser visto abaixo, o 'super' é usado no construtor das

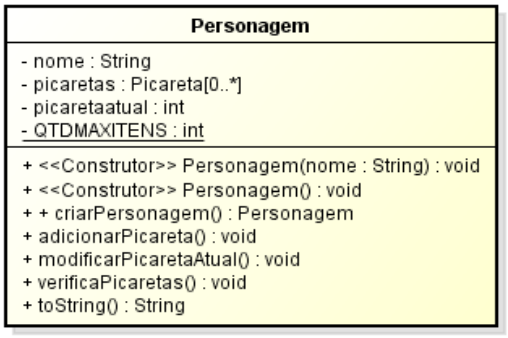
```
/**...4 linhas */
public class PicaretaMadeira extends Picareta{

    public PicaretaMadeira(){
        //através do construtor da superclasse define a durabilidade da picareta de madeira
        super(60);
    }
```

duas subclasses do projeto:

5.Diagrama de classes (obrigatório salvar também o png do diagrama no gitHub)







6.Todos os atributos devem ser inicializados. Fez validação de dados?

Dados inicializados no construtor da clase Personagem, a única classe em que isso é necessário:

```
//Construtor que recebe um nome como argumento.
public Personagem(String nome){
    if (Main.isDigito(nome))
        this.nome = "Padrao";
    else
        this.nome = nome;
    this.picaretas = new ArrayList<>();
    picaretaatual = -1;
}
//construtor vazio
public Personagem(){
    this.nome = "Padrao";
    this.picaretas = new ArrayList<>();
    picaretaatual = -1;
}
```

7.Dois construtores

Dois construtores na superclasse e nas subclasses:

```
//construtor
protected Picareta(int durabilidade){
    this.durabilidade = durabilidade;
    this.quebrado = false;
}
//construtor de cópia
protected Picareta(Picareta p){
    this.durabilidade = p.durabilidade;
    this.quebrado = p.quebrado;
}
```

```java
public PicaretaDiamante(){
    //através do construtor da superclasse define a durabilidade da picareta de madeira
    super(1563);
}
//construtor de cópia
public PicaretaDiamante(PicaretaDiamante Pd){
    super(Pd);
}
```

```java
public PicaretaMadeira(){
    //através do construtor da superclasse define a durabilidade da picareta de madeira
    super(60);
}
//construtor de cópia
public PicaretaMadeira(PicaretaMadeira Pm){
    super(Pm);
}
```

8.Um atributo static.

Variável enum de nome "Tipo" na classe Bloco.

```java
/**...4 linhas */
public class Bloco {
    //enum que define de quais tipos o bloco pode ser
    public enum Tipo{
        PEDRA, PEDREGULHO,TIJOLO, OURO, GELO,
    }
```

9.Um atributo const static

Atributo QTMAXITENS, que representa a quantidade máxima de itens que um personagem pode armazenar no seu inventário.

```java
public class Personagem {
    private String nome; //nome do personagem
    private ArrayList <Picareta> picaretas; //array de picaretas que o personagem guarda;
    private int picaretaatual; //picareta que o personagem está usando no momento.
    private static final int QTDMAXITENS = 5; //quantidade máxima de itens que o personagem pode guardar
```

Correta modelagem dos statics?

10.Um array

ArrayList de picaretas é usado na classe Personagem, para armazenar as picaretas criadas por ele.

```
public class Personagem {
    private String nome; //nome do personagem
    private ArrayList <Picareta> picaretas; //array de picaretas que o personagem guarda;
    private int picaretaatual; //picareta que o personagem está usando no momento.
    private static final int QTDMAXITENS = 5; //quantidade máxima de itens que o personagem pode guardar
```

Outro ArrayList, dessa vez de blocos, utilizado no main.

```
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        Random gerador = new Random();
        Personagem principal = new Personagem();
        ArrayList <Bloco> listabloco = new ArrayList<>();
        int opcao,nbloco;
```

11.Método static – deve ser chamado no main

Alguns métodos statics são criados e quase todos são utilizados no main:

isDigito():

```
//função que verifica se uma string possui dígito
public static boolean isDigito(String s){
    for (int i = 0;i < s.length();i++)
        if (Character.isDigit(s.charAt(i)))
            return true;

    return false;
}
```

geraArrayBlocos():

```
//gera um array de blocos aleatoriamente
public static void geraArrayBlocos(ArrayList <Bloco> array){
    Random gerador = new Random();
    int rand;

    //através de números aleatórios, adiciona automaticamente 20 elementos ao array
    //de blocos
    for (int i = 0;i < 15;i++){
        rand = gerador.nextInt(5);

        Bloco B = null;
        switch (rand){
            case 0:
                B = new Bloco(Bloco.Tipo.PEDRA);
                break;
            case 1:
                B = new Bloco(Bloco.Tipo.PEDREGULHO);
                break;
            case 2:
                B = new Bloco(Bloco.Tipo.TIJOLO);
                break;
            case 3:
                B = new Bloco(Bloco.Tipo.OURO);
                break;
            case 4:
                B = new Bloco(Bloco.Tipo.GELO);
                break;
        }
        array.add(B);
    }
}
```

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

```
//varre o array de blocos, eliminando o bloco que tiver sido quebrado por uma picareta
public static void verificaArrayBlocos(ArrayList <Bloco> array){
    for (int i = 0;i < array.size();i++){
        if (array.get(i).quebrado())
            array.remove(i);
    }
}
//imprime o array de blocos
public static void imprimeArrayBlocos(ArrayList <Bloco> array){
    for (int i = 0;i < array.size();i++)
        System.out.println((i + 1) + ". " + array.get(i) + ".");
}
//imprime a mensagem padrão do jogo
public static void mensagemPadrao(){
    System.out.println("Bem-vindo ao 'jogo' Minecraft -1.0 Beta!");
    System.out.println("Aperte enter para continuar: ");
    try {
        System.in.read();
    } catch (IOException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
}
//primeiro menu
public static void Menu1(){
    System.out.println("Escolha uma opção: ");
    System.out.println("1. Criar um personagem:");
    System.out.println("2. Sair do programa: ");
}
```

MenuPrincipal():

```
//menu principal do jogo
public static void MenuPrincipal(){
    System.out.println("Escolha uma opção abaixo:");
    System.out.println("1. Jogar");
    System.out.println("2. Adicionar uma picareta");
    System.out.println("3. Modificar a picareta atual");
    System.out.println("4. Sair do jogo");
}
```

12.Sobrescrita de método: chamar dentro do método da classe. Usar o @override

Sobrescrita usada no método *toString()* em todas as classes(fora o main):

```
@Override
public String toString(){
    String s = "Picareta de diamante";

    return s;
}
```

```
@Override
public String toString(){
    String s = "Picareta de madeira";

    return s;
}
```

```
@Override
public String toString(){
    String s = this.nome;
    return s;
}
```

```
@Override
public String toString(){
    String s = "Bloco de " + this.tipo;
    return s;
}
```

13.Usar Protected acessando diretamente os atributos na classe derivada

Protected usado na Superclasse Picareta:

```
public abstract class Picareta {

    /**
     * @param args the command line arguments
     */

    //durabilidade da picareta(quantos usos ela leva para quebrar).
    protected int durabilidade;
    //variável booleana que indica se a picareta está quebrada.
    protected boolean quebrado;
```

14.Usar suas classes no main

Objeto da classe Personagem e array de objetos da classe bloco declarados no main:

```
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        Random gerador = new Random();
        Personagem principal = new Personagem();
        ArrayList <Bloco> listabloco = new ArrayList<>();
        int opcao,nbloco;
```

Como um objeto da classe personagem possui um array de picaretas, a superclasse "Picareta" e as suas subclasses são utilizadas no programa principal através do objeto de Personagem "principal" criado. Um exemplo pode ser visto abaixo:

```
case 2:
    principal.adicionarPicareta();
    break;
case 3:
    principal.modificarPicaretaAtual();
    break;
```

Opcionais que garantem pontos extras:

Trabalhar com ENUM – Vide item 8.

Trabalhar com pacotes – Vide item 3.