```
package ArvoreRedBlack;

import TADStack.Stack;

public class RedBlackTree<T extends Comparable<T>> {
  private RedblackNode<T> root;

  public boolean isEmpty(){
    if(this.root == null){
      return true;
    } else {
      return false;

    }
  }

  public void insert(T info){
    RedblackNode<T> novo, aux;
    int retorno;
    novo = new RedblackNode<T>(info);
    if(this.isEmpty() == true){
      this.root = novo;
      novo.setPai(null);
      corrigirCor(novo);
    } else {
      aux = this.root;
      while(aux != null){
        retorno = info.compareTo(aux.getInfo());
        if(retorno == 0){
          System.out.println("Inserção cancelada, valor repetido");
          return;
        } else if(retorno < 0){
          //insercao a esquerda
          if(aux.getLeft() != null){
            aux = aux.getLeft();
          } else {
            aux.setLeft(novo);
            corrigirCor(novo);
          }
        }
      }
    }
  }

  private void corrigirCor(RedblackNode<T> no){
    if(this.root == no){
      no.setCor('B');
    } else if (no.getPai().getCor() == 'B'){
      //pai é black
      if(no.getTio().getCor() == 'B'){
        //se o tio for black
        no.setCor('R');
```

```
      } else if (no.getTio().getCor() == 'R'){
        //se o tio for red
        rotacao(no);
      }
    } else if(no.getPai().getCor() == 'R'){
      //pai é vermelho
      if(no.getTio().getCor() == 'R'){
        //tio eh vermelho
        //avo vira vermelho
        no.getAvo().setCor('R');
      } else if(no.getTio().getCor() == 'B'){
        //tio é preto
        rotacao(no);
      }
    }
  }

  private void rotacao(RedblackNode<T> no){
    RedblackNode<T> filhoDireita;
    RedblackNode<T> filhoEsquerda;
    if(no == no.getPai().getRight()){
      //rotacao a esquerda
      filhoDireita = no.getRight();
      no.setRight(filhoDireita.getLeft());

      if(filhoDireita.getLeft() != null){
        filhoDireita.getLeft().setPai(no);
      }
      filhoDireita.setPai(no.getPai());

      if(no.getPai() == null){
        this.root = filhoDireita;
      }else if(no == no.getPai().getLeft()){
        no.getPai().setLeft(filhoDireita);
      } else {
        no.getPai().setRight(filhoDireita);
      }

      filhoDireita.setLeft(no);
      no.setPai(filhoDireita);

    } else if(no == no.getPai().getLeft()){
      //rotacao a direita
      filhoEsquerda = no.getLeft();
      no.setLeft(filhoEsquerda.getRight());

      if(filhoEsquerda.getRight() != null){
        filhoEsquerda.getRight().setPai(no);
      }

      filhoEsquerda.setPai(no.getPai());
```

```
        if(no.getPai() == null){
            this.root = filhoEsquerda;
        }else if (no == no.getPai().getLeft()){
            no.getPai().setLeft(filhoEsquerda);
        } else {
            no.getPai().setRight(filhoEsquerda);
        }

        filhoEsquerda.setRight(no);
        no.setPai(filhoEsquerda);
    }
}

private RedblackNode<T> buscar(T value) {
    RedblackNode<T> aux;

    int retorno;
    if(this.isEmpty() == true) {
        return null;

    } else {
        aux = this.root;
        while(true) {
            retorno = value.compareTo(aux.getInfo());

            if(retorno == 0 && aux.isDisp() == true) {

                return aux;
            } else if (retorno < 0) {
                if(aux.getLeft() != null) {
                    aux = aux.getLeft();
                }
            } else if(retorno > 0) {
                if(aux.getRight() != null) {
                    aux = aux.getRight();
                }
            }


        }
    }
}

public void removerPreguicosa(T info){
    RedblackNode<T> retorno = buscar(info);

    retorno.setDisp(false);
}

public void passeioPorNivel(){
    RedblackNode<T> aux;
    if(this.isEmpty() == false) {
```

```
        QueueNode fila = new QueueNode();
        fila.enqueue(root);
        while(fila.isEmpty() == false) {
          aux = fila.dequeue();
          if(aux.getLeft() != null) {
            fila.enqueue(aux.getLeft());

          }
          if (aux.getRight() != null) {
            fila.enqueue(aux.getRight());
          }
          System.out.println(aux.getInfo());

        }
      } else {
        System.out.println("Arvore Vazia !");
      }

    }

    public void passeioEmOrdem(){
      Stack<RedblackNode<T>> pilha;
      RedblackNode<T> aux;
      if(this.isEmpty() == false ) {
        pilha = new Stack<RedblackNode<T>>();
        aux = this.root;
        while(pilha.isEmpty() == false || aux != null) {
          while(aux != null) {
            pilha.push(aux);
            aux = aux.getLeft();
          }
          aux = pilha.pop();
          System.out.println(aux.getInfo());
          aux = aux.getRight();
        }

      } else {
        System.out.println("Arvore Vazia!");
      }

    }

}
```

```
package ArvoreRedBlack;
class RedblackNode<T>{
  private RedblackNode<T> left;
  private RedblackNode<T> right;
  private T info;
  private RedblackNode<T> pai;
```

```
private char cor; // pode ser R (vermelho) ou B (preto)
private boolean disp;

public RedblackNode(T info){
   this.info = info;
   this.cor = 'R';
   this.disp = true;
}



public RedblackNode<T> getLeft() {
   return left;
}

public void setLeft(RedblackNode<T> left) {
   this.left = left;
}

public RedblackNode<T> getRight() {
   return right;
}

public void setRight(RedblackNode<T> right) {
   this.right = right;
}

public T getInfo() {
   return info;
}

public void setInfo(T info) {
   this.info = info;
}

public RedblackNode<T> getPai() {
   return pai;
}

public void setPai(RedblackNode<T> pai) {
   this.pai = pai;
}

public char getCor() {
   return cor;
}

public void setCor(char cor) {
   this.cor = cor;
}

public RedblackNode<T> getAvo(){
```

```
    return this.getPai().getPai();
  }

  public RedblackNode<T> getTio(){
    if(this.pai == this.getAvo().getLeft()){
      //pega o avo
      //se o pai for a esquerda do avo o tio é a direita do avo
      return this.getAvo().getRight();
    } else if (this.pai == this.getAvo().getRight()){
      return this.getAvo().getLeft();
    }
    // se o avo nao tiver nem esquerda nem direita ou avo for nulo
    return null;

  }

  public boolean isDisp() {
    return disp;
  }

  public void setDisp(boolean disp) {
    this.disp = disp;
  }



}
```