UNIVERSIDADE FEDERAL DO PARÁ
FACULDADE DE ENGENHARIA
DA COMPUTAÇÃO E TELECOMUNICAÇÕES

Alunos: Caio Bernardo Brasil – 202006840008
Daniel Cordeiro Campos – 202006840045
Gabriel Silva Ribeiro – 202007040021
Heitor Mesquita Anglada – 202006840018

1) Para implementar a função “desfazer”, utilizaria o tipo abstrato pilha, devido sua característica de remoção do último colocado. E para armazenar cliques do mouse, usaria filas por sua característica de transmitir o primeiro dado alterado.

2)
```
public boolean stackFull(){
//retorna verdadeiro se a ulltima posição ocupada na pilha for a de tamanho total da pilha
 return this.posicaoPilha == (this.pilha.length - 1);

}
```

3)
```
public void push(Object valor) {
   //Primiro ocorre a vericação pra ver se a pilha não está cheia para ai poder acrescentar algo
  if (this.posicaoPilha < this.pilha.length - 1)
     this.pilha[++posicaoPilha] = valor;

}
```

4)Apenas a primeira e a quarta sequência são possíveis, pois na segunda, após o pop no 8, só seria possível dar pop no 9 ou no 7, não no 1. Além desse, no terceiro, após dar pop no 3, só seria possível dar pop no 2 e não no 0.

5)O fragmento imprime o valor de 50 em binário, 110010. Então de modo geral, esse código serve para converter o valor em N de decimal para binário.

6)
```
class Fila{
  Pilha in = new Pilha();
  Pilha out = new Pilha();

  public void queue(Object iten){
     in.empilhar(iten);


  }

  public Object dequeue(){
```

```
    if(out.pilhaVazia()){
        while(!in.pilhaVazia()){//responsavel pela inversao de posições entre as duas pilhas
            out.empilhar(in.desempilhar());
        }

    }
    return out.desempilhar();

  }


}
```

7)
Caso n seja igual a 10, será retornado os 10 primeiros números da sequência de Fibonacci, 0 1 1 2 3 5 8 13 21 34. Então de modo geral, são retornados os n primeiros números da sequência de Fibonacci.

8)

9)

```
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Scanner in=new Scanner(System.in);

    //Insira o valor de N (ou seja, o número de colunas / rainhas)
    System.out.print("Entre com o valor ");
    int n = in.nextInt();

    //crie o objeto da classe NQueen e chame o método solve ()
    NQueen nQueen = new NQueen(n);
    nQueen.solve();
  }
}

class NQueen{
  int n;
  int board[][];
  boolean hasSolution = false;

  //construtor leva N como parâmetro
  NQueen(int n){
    this.n = n;
    board = new int[n][n];
  }

  //função exibir as configurações da placa
  private void printBoard(){
```

```
    for(int i=0; i<n; i++){
      for(int j=0; j<n; j++){
        if(board[i][j] == 1)
          System.out.print("* ");
        else
          System.out.print("- ");
      }
      System.out.println();
    }
    System.out.println("\n");
  }

  public void solve(){
    /*o solveColumn (int) é uma função recursiva, que verifica e posiciona a coluna das rainhas.
     Então, inicie a resolução da 1ª coluna */
    solveColumn(0);

    /* embora o solveColumn (int) produza as soluções possíveis,mas se nenhuma solução for
encontrada, envie a mensagem "No Soultion". */
    if(!hasSolution)
      System.out.println("No Solution");
  }

  //método de backtracking recursivo
  private void solveColumn(int col){
    /* se for além da última coluna, significa que uma solução (configuração) foi encontrada*/
    if(col == n){
      hasSolution= true;

      /* exibir a configuração da placa e voltar para encontrar mais soluções possíveis */
      printBoard();
      return;
    }

       //percorrer as células da coluna atual
    for(int i=0; i<n; i++){
      //verifique se a célula atual é segura para colocar uma rainha
      if(isValidPlace(i,col)){
        //se sim, coloque a rainha - mude o valor para 1 e mover para a próxima coluna
        board[i][col] = 1;
        solveColumn(col+1);
        //backtrack - redefinir para 0 significa remover rainha
        board[i][col] = 0;
      }

    }
  }

    //função verifica se a posição (r, c) é segura
  private boolean isValidPlace(int r, int c){
```

```
    //Verifique horizontalmente (esquerda)
    for(int j=c; j>=0; j--){
      if(board[r][j] == 1)
        return false;
    }

    //verifique a diagonal esquerda
    for(int i=r,j=c; i>=0 && j>=0; i--,j--){
      if(board[i][j] == 1)
        return false;
    }

    //verifique a diagonal direita
    for(int i=r,j=c; i<n && j>=0; i++,j--){
      if(board[i][j] == 1)
        return false;
    }

    //retornar verdadeiro porque é seguro
    return true;
  }
}
```