ESCOLA SUPERIOR DE TECNOLOGIA - IPS

ANO LETIVO 2015 / 2016

ENGENHARIA DE INFORMÁTICA

SISTEMAS OPERATIVOS. PROFª ROSSANA SANTOS

MANUAL TÉCNICO

MIGUEL FURTADO 120221006 TURMA G-02

Índice

Introdução

Este projeto foi desenvolvido para a unidade curricular de Sistemas Operativos da Licenciatura em Engenharia Informática Escola Superior de Tecnologia de Setúbal a pedido da docente Rossana Santos.

O projeto consiste no desenvolvimento de uma interface multimodal em que é operado o controlo de aviões, autocarros e comboios. Neste será simulado a chegada de aviões com passageiros em estes serão distribuídos pelos diferentes autocarros e comboios conforme o destino e transporte que lhes fora atribuído.

A implementação foi desenvolvida na linguagem de programação java e no software netbeans, foi usado ainda threads, sendo este um dos objetivos do trabalho e aplicados alguns dos conhecimentos dados nas aulas laboratoriais entre outros que foram adquiridos com base em pesquisa de informação relacionada com a matéria lecionada.

Justificação das opções de implementação

Na implementação da interface multimodal foram desenvolvidas as seguintes classes:

1. Grupo

A classe grupo foi criada com o propósito de guardar os diferentes grupos que cada avião contêm, desta forma, para além de guardar com mais facilidade a informação, o acesso à mesma foi fica também facilitado.

2. Hora

A classe hora tem como propósito definir o tempo em que toda a simulação vai decorrer e é uma classe fulcral pois todos os aviões, autocarros e comboios vão ter um horário e a partir deste é que os aviões, autocarros e comboios vão saber quando aterrar/estacionar. Para além disto é a partir da hora que é possível calcular os atrasos dos transportes e a que horas os grupos de passageiros entraram nos respetivos transportes.

3. Autocarro

A classe autocarro é runnable e vai estender da classe transporte e vai definir como é o objeto autocarro e como vai ser o comportamento deste. Com esta é possível controlar todos os movimentos do autocarro a partir do método run.

4. Avião

A classe avião é runnable e vai definir como é o objeto avião e como vai ser o comportamento deste. Com esta é possível controlar todos os movimentos do avião a partir do método run.

5. Comboio

A classe comboio é runnable e vai estender da classe transporte e vai definir como é o objeto comboio e como vai ser o comportamento deste. Com esta é possível controlar todos os movimentos do comboio a partir do método run.

6. Transporte

A classe transporte vai ser a classe que define os transportes autocarro e avião e todos os atributos principais e iguais que existem entre eles.

7. Aeroporto

A classe aeroporto vai ser a classe responsável pela criação e gestão dos aviões. Nesta decorrerá tudo o que está relacionado com as aterragens e partidas dos aviões.

8. TerminalFerroviario

A classe terminalFerroviario vai ser responsável pela criação e gestão dos comboios. Nesta decorrerá tudo o que está relacionado com as paragens e partidas dos comboios.

9. TerminalRodoviario

A classe terminalRodoviario vai ser responsável pela criação e gestão dos autocarros. Nesta decorrerá tudo o que está relacionado com as paragens e partidas dos comboios.

10. Central

A classe central vai ser a classe principal que vai ligar o aeoroporto, o terminalFerroviario e o terminalRodoviario. Na central o acesso a tudo irá ser facilitado e vai ser nesta irão decorrer as embarcações para os diferentes tipos de transporte.

Exemplo de funcionamento

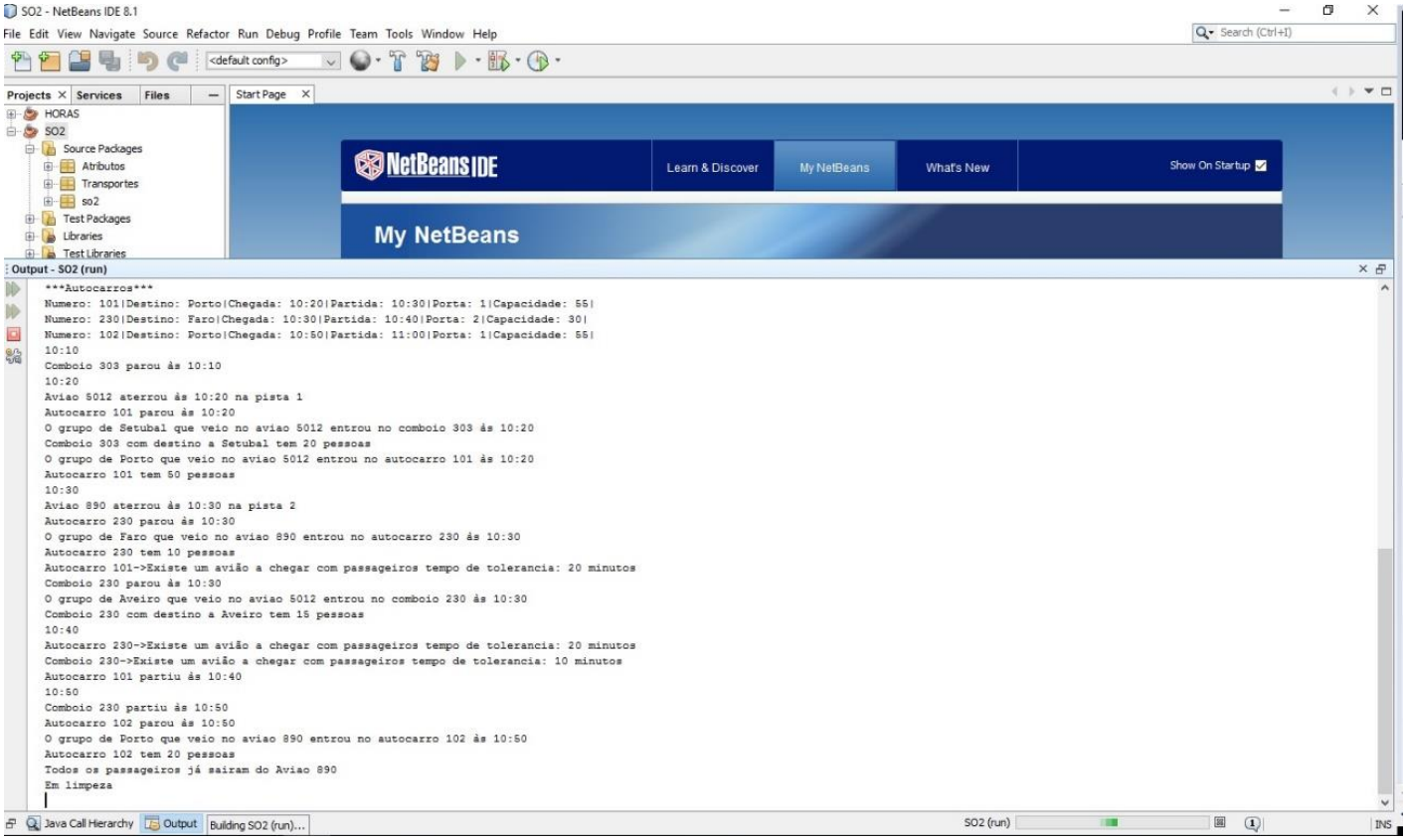

FIGURA 1 SIMULAÇÃO A DECORRER

Resultados das estatísticas

```
******INFORMAÇÃO DOS COMBOIOS******

Comboio 301 com destino a Coimbra chegou às 12:40 com um atraso de 140 minutos partiu às 12:50 com um atraso de 140 minutos e com 40 passageiros
Grupos que embarcaram:
Grupo de 20 pessoas que chegou no avião 1027 entrou no comboio às 12:40 e esperou 100 minutos
Grupo de 20 pessoas que chegou no avião 5012 entrou no comboio às 12:40 e esperou 140 minutos

Comboio 230 com destino a Aveiro chegou às 10:30 a horas partiu às 10:50 com um atraso de 10 minutos e com 15 passageiros
Grupos que embarcaram:
Grupo de 15 pessoas que chegou no avião 5012 entrou no comboio às 10:30 e esperou 10 minutos

Comboio 302 com destino a Porto chegou às 12:30 com um atraso de 100 minutos partiu às 12:40 com um atraso de 100 minutos e com 0 passageiros
Não teve embarcações

Comboio 303 com destino a Setubal chegou às 10:10 a horas partiu às 12:30 com um atraso de 10 minutos e com 30 passageiros
Grupos que embarcaram:
Grupo de 20 pessoas que chegou no avião 5012 entrou no comboio às 10:20 e esperou 0 minutos
Grupo de 10 pessoas que chegou no avião 1027 entrou no comboio às 11:00 e esperou 0 minutos
****************************************

******INFORMAÇÃO DOS Autocarros******

Autocarro 101 com destino a Porto chegou às 10:20 a horas partiu às 10:40 com um atraso de 10 minutos e com 50 passageiros
Grupos que embarcaram:
Grupo de 50 pessoas que chegou no avião 5012 entrou no comboio às 10:20 e esperou 0 minutos

Autocarro 230 com destino a Faro chegou às 10:30 a horas partiu às 10:50 com um atraso de 10 minutos e com 10 passageiros
Grupos que embarcaram:
Grupo de 10 pessoas que chegou no avião 890 entrou no comboio às 10:30 e esperou 0 minutos

Autocarro 102 com destino a Porto chegou às 10:50 a horas partiu às 11:10 com um atraso de 10 minutos e com 40 passageiros
Grupos que embarcaram:
Grupo de 20 pessoas que chegou no avião 890 entrou no comboio às 10:50 e esperou 20 minutos
Grupo de 20 pessoas que chegou no avião 1027 entrou no comboio às 11:00 e esperou 0 minutos
****************************************
```

Métodos Mais Importantes

1. Método run da classe Autocarro

O método run é usado para controlar o comportamento das threads do tipo autocarro. Inicialmente este encontra-se todo dentro de um while(true) para que apenas seja interrompido caso a simulação seja interrompida.

Esta foi usada dentro de um switch que vai buscar o terminal corresponde ao autocarro onde é suposto estacionar. Como existem 3 plataformas foram usados 3 cases, mas apenas será descrito o funcionamento do case 1.

Ao entrar no case 1, o autocarro primeiro irá ficar se já estacionou e se o terminal se encontra livre, caso não se encontre livre ou este já tenho estacionado uma vez, o mesmo irá ficar em wait. De seguida a thread ira confirmar, com base na sua hora de chegada, se pode estacionar, caso não posso a thread fica a dormir e irá confirmar a cada 100 milissegundos se pode já pode estacionar. Ao passar esta restrição a mesma irá guardar que já estacionou uma vez e irá aceder ao método chegadaTerminal1 que se encontra na classe terminalRodoviario e para alterar alguns dados que indicaram que o mesmo estacionou. Após isto a thread se ainda não puder partir irá entrar num while onde a cada 100 milissegundos vai confirmar se já pode partir, ao sair deste while irá confirmar se existe algum avião com passageiros por aterrar, caso haja a thread espera 200 milissegundos que equivalente aos 20minutos de tolerância pedidos no enunciado do projeto. Por fim a thread já fez o que devia e vai partir,

logo acedo ao método partida e altera alguns dados para que esta fique indicada de que partiu e altera o atributo das horas de partida do objeto.

```
@Override
   public void run() {
      while (true) {
         switch (getTerminal()) {
            case 1:
               if (getChegadaDiaria() == 0 && terminalRodoviario.terminalLivre(0)) {//Ainda
não estacionou hoje e o terminal está livre
                  if (!getChegadaPrevista().isAfter(getCentral().getHoras())) {
                     setChegadaDiaria(1);//já chegou uma vez hoje

                     terminalRodoviario.chegadaTerminal1(this);//chega ao terminal

                     while (getPartidaPrevista().isAfter(getCentral().getHoras())) {//enquanto
não puder partir dorme
                        terminalRodoviario.esperarAtePartir();
                     }

                     if (getCentral().devoEsperar(this.getDestino())) {//se existir um aviao a vir
dorme
                        terminalRodoviario.tolerancia(this);
                     }

                     terminalRodoviario.partida(0, this);//parte do terminal
                     setPartida(getCentral().getHoras());

                  } else {
                     terminalRodoviario.esperarAtePartir();
                  }
               } else {//Não está livre o terminal ou já estacionou uma vez então espera
                  terminalRodoviario.esperarAteLivre();
               }

               break;
            case 2:
               if (getChegadaDiaria() == 0 && terminalRodoviario.terminalLivre(1)) {
                  if (!getChegadaPrevista().isAfter(getCentral().getHoras())) {
                     setChegadaDiaria(1);

                     terminalRodoviario.chegadaTerminal2(this);

                     while (getPartidaPrevista().isAfter(getCentral().getHoras())) {
                        terminalRodoviario.esperarAtePartir();
                     }

                     if (getCentral().devoEsperar(this.getDestino())) {
                        terminalRodoviario.tolerancia(this);
                     }
```

```
                terminalRodoviario.partida(1, this);

              } else {
                terminalRodoviario.esperaParaEstacionar();
              }
            } else {
              terminalRodoviario.esperarAteLivre();
            }

            break;
          case 3:
            if (getChegadaDiaria() == 0 && terminalRodoviario.terminalLivre(2)) {
              if (!getChegadaPrevista().isAfter(getCentral().getHoras())) {
                setChegadaDiaria(1);

                terminalRodoviario.chegadaTerminal3(this);

                while (getPartidaPrevista().isAfter(getCentral().getHoras())) {
                  terminalRodoviario.esperarAtePartir();
                }

                if (getCentral().devoEsperar(this.getDestino())) {
                  terminalRodoviario.tolerancia(this);
                }

                terminalRodoviario.partida(2, this);

              } else {
                terminalRodoviario.esperaParaEstacionar();
              }
            } else {
              terminalRodoviario.esperarAteLivre();
            }
            break;
        }
      }
    }
```

2. Método run da classe Avião

O método run foi usado para definir o comportamento das threads do tipo avião. Para este foi usado um while(true) para que apenas seja interrompido o comportamento destas caso a simulação seja interrompida. Inicialmente é usado um switch para saber qual o terminal que a thread tem que aceder com base no que atributo terminal que corresponde a cada avião.

De seguida o avião acede a uma restrição em que verifica se já aterrou, pois só pode aterrar uma vez por dia e se a pista de aterragem está livre. Caso não esteja este fica parado num wait. Após passar esta verificação, vai confirmar se é a sua hora de chegadaPrevista caso seja ou já tenha passado o tempo, indica que já aterrou uma vez, guarda a hora real de chegada e aterra, caso não seja fica a dormir e a cada 100milissegundos vai confirmar se já pode aterrar. Após aterrar vai confirmar se todos os grupos existentes no avião já embarcaram, caso seja falso e ainda faltem grupos por embarcar, irá ser visto os grupos que faltam sair e se o transporte que estes precisam de apanhar é autocarro ou comboio e ainda se existem algum desses transportes estacionadas nas respetivas paragens. Caso não haja fica à espera até haver transportes.

Após todos os grupos saírem do avião este vai ser limpo durante 30minutos e após a limpeza vai levantar voo.

```
@Override
   public void run() {
      while (true) {
         switch (this.terminal) {
            case 1:
               if (!aeroporto.jaAterrou(this) && aeroporto.pistaLivre(0))//se ainda não
aterrou 1 vez e a pista está livre aterra
                  {
                     if ((!getChegadaPrevista().isAfter(central.getHoras()))) {
                        setAterragemDiaria(1);//definir que já aterrou uma vez
                        setChegada(central.getHoras());//definir a hora de chegada
verdadeira

                        aeroporto.aterra1(this);//aterrar

                        while (this.todosEmbarcaram == false &&
aeroporto.gruposFaltam(grupos) != 0) {//enquanto houver grupos por embarcar
continua a tentar embarca-los
                           for (Grupo g : grupos) {
                              if (g.getTransporte().equals("autocarro") &&
central.getTerminalRodoviario().getAutocarroParado() != null) {// se o grupo tiver
como transporte autocarro e houver autocarros parados
                                 central.embarcarAutocarro(g, this);//embarcar no autocarro

                           } else if (g.getTransporte().equals("comboio") &&
central.getTerminalFerroviario().getComboioParado() != null) {//se o grupos tiver
como transporte comboio e houver comboios parados
```

```
                        central.embarcarComboio(g, this);//embarcar no comboio

                    } else {
                        aeroporto.esperaPorComboiosOuAutocarros();//metodo
para dizer ao aviao caso haja grupos para esperar por comboios ou autucarros
                    }
                }
                aeroporto.gruposEmbarcaram(grupos, this);// metodo que vai
confirmar se todos os grupos já embarcaram
            }

            this.aeroporto.LimparAviao();//metodo para limpar o aviao apos
todos os grupos terem desembarcado
            this.aeroporto.partida(0, this);//metodo para o aviao puder partir
            this.partida = this.central.getHoras();//define a hora de partida
          } else {
            aeroporto.esperaPorHoraDeAterrar(); // caso não seja a hora dele
de aterrar espera
          }
        } else {
          aeroporto.esperaPorPistaLivre();//caso a pista não esteja livre espera
        }
        break;
      case 2:
        if (!aeroporto.jaAterrou(this) && aeroporto.pistaLivre(1)) {
          if (!getChegadaPrevista().isAfter(central.getHoras())) {
            setAterragemDiaria(1);
            setChegada(central.getHoras());
            aeroporto.aterra2(this);

            while (this.todosEmbarcaram == false &&
aeroporto.gruposFaltam(grupos) != 0) {
              for (Grupo g : grupos) {
                if (g.getTransporte().equals("autocarro") &&
central.getTerminalRodoviario().getAutocarroParado() != null) {
                  central.embarcarAutocarro(g, this);

                } else if (g.getTransporte().equals("comboio") &&
central.getTerminalFerroviario().getComboioParado() != null) {
                  central.embarcarComboio(g, this);

                } else {
                  aeroporto.esperaPorComboiosOuAutocarros();
                }
              }

              aeroporto.gruposEmbarcaram(grupos, this);
```

```
                }

                this.aeroporto.LimparAviao();
                aeroporto.partida(1, this);
                this.partida = central.getHoras();
              } else {
                aeroporto.esperaPorHoraDeAterrar();
              }
            } else {
              aeroporto.esperaPorPistaLivre();
            }
            break;
          case 3:
            if (!aeroporto.jaAterrou(this) && aeroporto.pistaLivre(2)) {
              if (!getChegadaPrevista().isAfter(central.getHoras())) {
                setAterragemDiaria(1);
                setChegada(central.getHoras());
                aeroporto.aterra3(this);

                while (this.todosEmbarcaram == false &&
aeroporto.gruposFaltam(grupos) != 0) {
                  for (Grupo g : grupos) {
                    if (g.getTransporte().equals("autocarro") &&
central.getTerminalRodoviario().getAutocarroParado() != null) {
                      central.embarcarAutocarro(g, this);

                    } else if (g.getTransporte().equals("comboio") &&
central.getTerminalFerroviario().getComboioParado() != null) {
                      central.embarcarComboio(g, this);

                    } else {
                      aeroporto.esperaPorComboiosOuAutocarros();
                    }
                  }

                  aeroporto.gruposEmbarcaram(grupos, this);
                }

                this.aeroporto.LimparAviao();
                aeroporto.partida(2, this);
                this.partida = central.getHoras();
              } else {
                aeroporto.esperaPorHoraDeAterrar();
              }
            } else {
              aeroporto.esperaPorPistaLivre();
            }
```

```
                break;
            }
        }
    }
```

3. Método run da classe Comboio

O método run foi usado para definir o comportamento da thread comboio. Este funciona exatamente da mesma maneira que o método run da classe autocarro. Sendo que as diferenças é que os métodos que adormecem as threads ou as deixam em wait estão na classe TerminalFerroviario e não na classe TerminalRodoviario.

```
@Override
  public void run() {
    while (true) {
      switch (getTerminal()) {
        case 1:
          if (getChegadaDiaria() == 0 && terminalFerroviario.terminalLivre(0))//se
ainda nãoe estacioun uma vez hoje e se o terminal está livre
            if (!getChegadaPrevista().isAfter(getCentral().getHoras())) {
              setChegadaDiaria(1);//define que já estacionou uma vez hoje

              terminalFerroviario.chegadaTerminal1(this); //metodo que vai
definir que o comboio estacioun

              while (getPartidaPrevista().isAfter(getCentral().getHoras())) {
                terminalFerroviario.esperarAtePartir();//enquanto não for a hora
de partir espera
              }

              if (getCentral().devoEsperar(this.getDestino()))//se vem um aviao
com passageiros que vão apanhar este autocarro espera
                terminalFerroviario.tolerancia(this);//espera tempo de tolerancia

              terminalFerroviario.partida(0, this);//metodo de partida do
comboio da estação

            } else
              terminalFerroviario.esperarParaEstacionar();//espera pela hora
para estacionar
          else
            terminalFerroviario.esperaAteLivre();//espera até a plataforma estar
livre
          break;

        case 2:
          if (getChegadaDiaria() == 0 && terminalFerroviario.terminalLivre(1))
            if ((!getChegadaPrevista().isAfter(getCentral().getHoras()))) {
```

```
                    setChegadaDiaria(1);

                    this.terminalFerroviario.chegadaTerminal2(this);

                    while (getPartidaPrevista().isAfter(getCentral().getHoras())) {
                      terminalFerroviario.esperarAtePartir();
                    }

                    if (getCentral().devoEsperar(this.getDestino()))
                      terminalFerroviario.tolerancia(this);

                    this.terminalFerroviario.partida(1, this);

                  } else
                    terminalFerroviario.esperarParaEstacionar();
                else
                  terminalFerroviario.esperaAteLivre();
                break;
            }
          }
        }
```

Análise das limitações

Algumas das limitações sentidas, foram apenas no início do desenvolvimento. Deparei-me com a falta de informação adquirida ao começar, mas com o desenrolar do projeto e após ultrapassar algumas dificuldades para que obtivesse um bom funcionamento das threads, foi possível ultrapassa-las. Claro que uma grande ajuda foi obtida com pesquisas na internet onde aprendi muito com casos semelhantes de dificuldades encontradas.

Código Fonte

Classe Grupos

```
public class Grupo {

  private int numeroPessoas;
  private String transporte;
  private String destino;
  private LocalTime horaEntrada;
  private boolean embarcou;
  private LocalTime horaChegada;
  private int numeroAviao;
  private long tempoEspera;

  public Grupo(int numeroPessoas, String transporte, String destino, int numeroAviao) {
    this.numeroPessoas = numeroPessoas;
    this.transporte = transporte;
    this.destino = destino;
    this.horaEntrada = null;
```

```
    this.embarcou = false;
    this.horaChegada = null;
    this.numeroAviao = numeroAviao;
    this.tempoEspera = 0;
}

public int getNumeroPessoas() {
    return numeroPessoas;
}

public void setNumeroPessoas(int numeroPessoas) {
    this.numeroPessoas = numeroPessoas;
}

public String getTransporte() {
    return transporte;
}

public void setTransporte(String transporte) {
    this.transporte = transporte;
}

public String getDestino() {
    return destino;
}

public void setDestino(String destino) {
    this.destino = destino;
}

public LocalTime getHoraEntrada() {
    return horaEntrada;
}

public synchronized void setHoraEntrada(LocalTime horaEntrada) {
    this.horaEntrada = horaEntrada;
}

public boolean isEmbarcou() {
    return embarcou;
}

public synchronized void setEmbarcou(boolean embarcou) {
    this.embarcou = embarcou;
}

public LocalTime getHoraChegada() {
```

```
        return horaChegada;
    }

    public synchronized void setHoraChegada(LocalTime horaChegada) {
        this.horaChegada = horaChegada;
    }

    public int getNumeroAviao() {
        return numeroAviao;
    }

    public void setNumeroAviao(int numeroAviao) {
        this.numeroAviao = numeroAviao;
    }

    public long getTempoEspera() {
        return tempoEspera;
    }

    public synchronized void setTempoEspera(long tempoEspera) {
        this.tempoEspera = tempoEspera;
    }
}
```

Classe Horas

```
public class Hora extends Thread {

    private LocalTime time;
    private Central central;

    public Hora(LocalTime time, Central central) {
        this.time = time;
        this.central = central;
    }

    public LocalTime getTime() {
        return time;
    }

    public LocalTime horaInicial() {
        return time;
    }

    public Central getCentral() {
        return central;
    }
```

```
  /**
   * Método que vai diz qual o funcionamente da thread horas
   */
  @Override
  public void run() {
    while (true) {
      try {
        Thread.sleep(100);
      } catch (InterruptedException ex) {
        Logger.getLogger(Hora.class.getName()).log(Level.SEVERE, null, ex);
      }

      time = time.plusMinutes(10);
      System.out.println(time);

      if (time.getHour() == 0 && time.getMinute() == 0){
        System.out.println(central.toString());
        System.exit(0);
      }

    }

  }

  @Override
  public String toString() {
    return getTime().toString();
  }
}
```

Classe Autocarro

```
public class Autocarro extends Transporte implements Runnable {

  private TerminalRodoviario terminalRodoviario;
  private ArrayList<Grupo> gruposAutocarro;

  public Autocarro(Central central, TerminalRodoviario terminal) {
    super(central);
    this.terminalRodoviario = terminal;
    this.gruposAutocarro = new ArrayList<>();
  }

  public Autocarro(int numero, String destino, int terminal, LocalTime chegadaPrevista,
      LocalTime partidaPrevista, LocalTime horaActual, int capacidade, int lugaresOcupados,
Central central, TerminalRodoviario terminalRodoviario) {
```

```
    super(numero, destino, terminal, chegadaPrevista, partidaPrevista, horaActual, capacidade,
            lugaresOcupados, central);
    this.terminalRodoviario = terminalRodoviario;
  }

  public TerminalRodoviario getTerminalRodoviario() {
    return terminalRodoviario;
  }

  public void setTerminalRodoviario(TerminalRodoviario terminalRodoviario) {
    this.terminalRodoviario = terminalRodoviario;
  }

  public ArrayList<Grupo> getGruposAutocarro() {
    return gruposAutocarro;
  }

  public void setGruposAutocarro(ArrayList<Grupo> gruposAutocarro) {
    this.gruposAutocarro = gruposAutocarro;
  }
  /**
   * Método que vai controlar o funcionamente das várias thread autocarros
   */
  @Override
  public void run() {
    while (true) {
      switch (getTerminal()) {
        case 1:
          if (getChegadaDiaria() == 0 && terminalRodoviario.terminalLivre(0)) {//Ainda não estacionou hoje e o terminal está livre
            if (!getChegadaPrevista().isAfter(getCentral().getHoras())) {
              setChegadaDiaria(1);//já chegou uma vez hoje

              terminalRodoviario.chegadaTerminal1(this);//chega ao terminal

              while (getPartidaPrevista().isAfter(getCentral().getHoras())) {//enquanto não puder partir dorme
                terminalRodoviario.esperarAtePartir();
              }

              if (getCentral().devoEsperar(this.getDestino())) {//se existir um aviao a vir dorme
                terminalRodoviario.tolerancia(this);
              }

              terminalRodoviario.partida(0, this);//parte do terminal
              setPartida(getCentral().getHoras());
```

```
      } else {
        terminalRodoviario.esperarAtePartir();
      }
    } else {//Não está livre o terminal ou já estacionou uma vez então espera
      terminalRodoviario.esperarAteLlivre();
    }

    break;
  case 2:
    if (getChegadaDiaria() == 0 && terminalRodoviario.terminalLivre(1)) {
      if (!getChegadaPrevista().isAfter(getCentral().getHoras())) {
        setChegadaDiaria(1);

        terminalRodoviario.chegadaTerminal2(this);

        while (getPartidaPrevista().isAfter(getCentral().getHoras())) {
          terminalRodoviario.esperarAtePartir();
        }

        if (getCentral().devoEsperar(this.getDestino())) {
          terminalRodoviario.tolerancia(this);
        }

        terminalRodoviario.partida(1, this);

      } else {
        terminalRodoviario.esperaParaEstacionar();
      }
    } else {
      terminalRodoviario.esperarAteLivre();
    }

    break;
  case 3:
    if (getChegadaDiaria() == 0 && terminalRodoviario.terminalLivre(2)) {
      if (!getChegadaPrevista().isAfter(getCentral().getHoras())) {
        setChegadaDiaria(1);

        terminalRodoviario.chegadaTerminal3(this);

        while (getPartidaPrevista().isAfter(getCentral().getHoras())) {
          terminalRodoviario.esperarAtePartir();
        }

        if (getCentral().devoEsperar(this.getDestino())) {
          terminalRodoviario.tolerancia(this);
```

```
                }

                terminalRodoviario.partida(2, this);

            } else {
                terminalRodoviario.esperaParaEstacionar();
            }
        } else {
            terminalRodoviario.esperarAteLivre();
        }
        break;
    }
  }
}

@Override
public String toString() {
  StringBuilder str = new StringBuilder();

  str.append(super.toString()).append("|");

  return str.toString();

}

}
```

Classe Aviao

```
public class Aviao implements Runnable {

  private int numero;
  private String origem;
  private int terminal;
  private LocalTime chegadaPrevista;
  private LocalTime partidaPrevista;
  private LocalTime chegada;
  private LocalTime partida;
  private Central central;
  private ArrayList<Grupo> grupos;
  private Aeroporto aeroporto;
  private int aterragemDiaria;
  private boolean todosEmbarcaram;
  private boolean descolou;

  public Aviao(Aeroporto aeroporto, Central central) {
    this.aeroporto = aeroporto;
    this.grupos = new ArrayList<>();
```

```
        this.central = central;
        this.chegada = null;
        this.partida = null;
        this.chegadaPrevista = null;
        this.aterragemDiaria = 0;
        this.todosEmbarcaram = false;
        this.descolou = false;

    }

    public Aviao(int numero, String origem, int terminal,
            LocalTime chegadaPrevista, LocalTime partidaPrevista, LocalTime chegada,
            LocalTime partida, LocalTime horaActual, Central central, Grupo[] grupos, Aeroporto
aeroporto) {
        this.numero = numero;
        this.origem = origem;
        this.terminal = terminal;
        this.chegadaPrevista = null;
        this.partidaPrevista = null;
        this.chegada = null;
        this.partida = null;
        this.central = central;
        this.aeroporto = aeroporto;
        this.grupos = new ArrayList<>();
        this.aterragemDiaria = 0;
        this.todosEmbarcaram = false;
        this.descolou = false;

    }

    public ArrayList<Grupo> getGrupos() {
        return grupos;
    }

    public void setGrupos(ArrayList<Grupo> grupos) {
        this.grupos = grupos;
    }

    public Aeroporto getAeroporto() {
        return aeroporto;
    }

    public void setAeroporto(Aeroporto aeroporto) {
        this.aeroporto = aeroporto;
    }

    public int getNumero() {
```

```
    return numero;
}

public void setNumero(int numero) {
    this.numero = numero;
}

public String getOrigem() {
    return origem;
}

public void setOrigem(String origem) {
    this.origem = origem;
}

public int getTerminal() {
    return terminal;
}

public void setTerminal(int terminal) {
    this.terminal = terminal;
}

public Central getCentral() {
    return central;
}

public void setCentral(Central central) {
    this.central = central;
}

public int getAterragemDiaria() {
    return aterragemDiaria;
}

public synchronized void setAterragemDiaria(int aterragemDiaria) {
    this.aterragemDiaria = aterragemDiaria;
}

public LocalTime getChegadaPrevista() {
    return chegadaPrevista;
}

public void setChegadaPrevista(LocalTime chegadaPrevista) {
    this.chegadaPrevista = chegadaPrevista;
}
```

```
public LocalTime getPartidaPrevista() {
   return partidaPrevista;
}

public void setPartidaPrevista(LocalTime partidaPrevista) {
   this.partidaPrevista = partidaPrevista;
}

public LocalTime getChegada() {
   return chegada;
}

public void setChegada(LocalTime chegada) {
   this.chegada = chegada;
}

public LocalTime getPartida() {
   return partida;
}

public void setPartida(LocalTime partida) {
   this.partida = partida;
}

public boolean isTodosEmbarcaram() {
   return todosEmbarcaram;
}

public void setTodosEmbarcaram(boolean todosEmbarcaram) {
   this.todosEmbarcaram = todosEmbarcaram;
}

public boolean isDescolou() {
   return descolou;
}

public void setDescolou(boolean descolou) {
   this.descolou = descolou;
}

/**
 * Método run que irá determinar o funcionamento dos avioes
 */
@Override
public void run() {
   while (true) {
      switch (this.terminal) {
```

```
        case 1:
          if (!aeroporto.jaAterrou(this) && aeroporto.pistaLivre(0))//se ainda não aterrou 1 vez e a pista está livre aterra
            {
              if ((!getChegadaPrevista().isAfter(central.getHoras()))) {
                setAterragemDiaria(1);//definir que já aterrou uma vez
                setChegada(central.getHoras());//definir a hora de chegada verdadeira

                aeroporto.aterra1(this);//aterrar

                while (this.todosEmbarcaram == false && aeroporto.gruposFaltam(grupos) != 0) {//enquanto houver grupos por embarcar continua a tentar embarca-los
                  for (Grupo g : grupos) {
                    if (g.getTransporte().equals("autocarro") && central.getTerminalRodoviario().getAutocarroParado() != null) {// se o grupo tiver como transporte autocarro e houver autocarros parados
                      central.embarcarAutocarro(g, this);//embarcar no autocarro

                    } else if (g.getTransporte().equals("comboio") && central.getTerminalFerroviario().getComboioParado() != null) {//se o grupos tiver como transporte comboio e houver comboios parados
                      central.embarcarComboio(g, this);//embarcar no comboio

                    } else {
                      aeroporto.esperaPorComboiosOuAutocarros();//metodo para dizer ao aviao caso haja grupos para esperar por comboios ou autucarros
                    }
                  }
                  aeroporto.gruposEmbarcaram(grupos, this);// metodo que vai confirmar se todos os grupos já embarcaram
                }

                this.aeroporto.LimparAviao();//metodo para limpar o aviao apos todos os grupos terem desembarcado
                this.aeroporto.partida(0, this);//metodo para o aviao puder partir
                this.partida = this.central.getHoras();//define a hora de partida
              } else {
                aeroporto.esperaPorHoraDeAterrar(); // caso não seja a hora dele de aterrar espera
              }
            } else {
              aeroporto.esperaPorPistaLivre();//caso a pista não esteja livre espera
            }
            break;
          case 2:
            if (!aeroporto.jaAterrou(this) && aeroporto.pistaLivre(1)) {
              if (!getChegadaPrevista().isAfter(central.getHoras())) {
```

```
                setAterragemDiaria(1);
                setChegada(central.getHoras());
                aeroporto.aterra2(this);

                while (this.todosEmbarcaram == false && aeroporto.gruposFaltam(grupos) != 0)
{
                  for (Grupo g : grupos) {
                    if (g.getTransporte().equals("autocarro") &&
central.getTerminalRodoviario().getAutocarroParado() != null) {
                      central.embarcarAutocarro(g, this);

                    } else if (g.getTransporte().equals("comboio") &&
central.getTerminalFerroviario().getComboioParado() != null) {
                      central.embarcarComboio(g, this);

                    } else {
                      aeroporto.esperaPorComboiosOuAutocarros();
                    }
                  }

                  aeroporto.gruposEmbarcaram(grupos, this);
                }

                this.aeroporto.LimparAviao();
                aeroporto.partida(1, this);
                this.partida = central.getHoras();
              } else {
                aeroporto.esperaPorHoraDeAterrar();
              }
            } else {
              aeroporto.esperaPorPistaLivre();
            }
            break;
          case 3:
            if (!aeroporto.jaAterrou(this) && aeroporto.pistaLivre(2)) {
              if (!getChegadaPrevista().isAfter(central.getHoras())) {
                setAterragemDiaria(1);
                setChegada(central.getHoras());
                aeroporto.aterra3(this);

                while (this.todosEmbarcaram == false && aeroporto.gruposFaltam(grupos) != 0)
{
                  for (Grupo g : grupos) {
                    if (g.getTransporte().equals("autocarro") &&
central.getTerminalRodoviario().getAutocarroParado() != null) {
                      central.embarcarAutocarro(g, this);
```

```
                    } else if (g.getTransporte().equals("comboio") &&
central.getTerminalFerroviario().getComboioParado() != null) {
                        central.embarcarComboio(g, this);

                    } else {
                        aeroporto.esperaPorComboiosOuAutocarros();
                    }
                }

                aeroporto.gruposEmbarcaram(grupos, this);
            }

            this.aeroporto.LimparAviao();
            aeroporto.partida(2, this);
            this.partida = central.getHoras();
          } else {
            aeroporto.esperaPorHoraDeAterrar();
          }
        } else {
          aeroporto.esperaPorPistaLivre();
        }
        break;
    }
  }
  }

  @Override
  public String toString() {
    StringBuilder str = new StringBuilder();

    str.append("Numero: ").append(getNumero()).append("|");
    str.append("Origem: ").append(getOrigem()).append("|");
    str.append("Chegada: ").append(getChegadaPrevista()).append("|");
    str.append("Porta: ").append(getTerminal());

    for (int i = 0; i < grupos.size(); i++) {

      str.append("\n");
      str.append("Numero                                                    Pessoas:
").append(getGrupos().get(i).getNumeroPessoas()).append("|");
      str.append("Transporte: ").append(getGrupos().get(i).getTransporte()).append("|");
      str.append("Destino: ").append(getGrupos().get(i).getDestino());

    }
    str.append("\n");

    return str.toString();
```

```
    }

}
```

Classe Comboio

```
        public class Comboio extends Transporte implements Runnable {

    private TerminalFerroviario terminalFerroviario;
    private ArrayList<Grupo> gruposComboio;

    public Comboio(Central central, TerminalFerroviario terminalFerroviario) {
        super(central);
        this.terminalFerroviario = terminalFerroviario;
        this.gruposComboio = new ArrayList<>();
    }

    public Comboio(int numero, String destino, int terminal, LocalTime chegadaPrevista,
            LocalTime partidaPrevista, LocalTime horaActual, int capacidade, int lugaresOcupados,
Central central,
            TerminalFerroviario terminalFerroviario) {

        super(numero, destino, terminal, chegadaPrevista, partidaPrevista, horaActual,
                capacidade, lugaresOcupados, central);

        this.terminalFerroviario = terminalFerroviario;

    }

    public TerminalFerroviario getTerminalFerroviario() {
        return terminalFerroviario;
    }

    public void setTerminalFerroviario(TerminalFerroviario terminalFerroviario) {
        this.terminalFerroviario = terminalFerroviario;
    }

    public ArrayList<Grupo> getGruposComboio() {
        return gruposComboio;
    }

    public void setGruposComboio(ArrayList<Grupo> gruposComboio) {
        this.gruposComboio = gruposComboio;
    }
    /**
     * Método  que vai determinar o funcionamento das threads comboio
     */
    @Override
```

```
  public void run() {
    while (true) {
      switch (getTerminal()) {
        case 1:
          if (getChegadaDiaria() == 0 && terminalFerroviario.terminalLivre(0))//se ainda nãoe
estacioun uma vez hoje e se o terminal está livre
            if (!getChegadaPrevista().isAfter(getCentral().getHoras())) {
              setChegadaDiaria(1);//define que já estacionou uma vez hoje

              terminalFerroviario.chegadaTerminal1(this); //metodo que vai definir que o
comboio estacioun

              while (getPartidaPrevista().isAfter(getCentral().getHoras())) {
                terminalFerroviario.esperarAtePartir();//enquanto não for a hora de partir
espera
              }

              if (getCentral().devoEsperar(this.getDestino()))//se vem um aviao com
passageiros que vão apanhar este autocarro espera
                terminalFerroviario.tolerancia(this);//espera tempo de tolerancia

              terminalFerroviario.partida(0, this);//metodo de partida do comboio da estação

            } else
              terminalFerroviario.esperarParaEstacionar();//espera pela hora para estacionar
          else
            terminalFerroviario.esperaAteLivre();//espera até a plataforma estar livre
          break;

        case 2:
          if (getChegadaDiaria() == 0 && terminalFerroviario.terminalLivre(1))
            if ((!getChegadaPrevista().isAfter(getCentral().getHoras())) {
              setChegadaDiaria(1);

              this.terminalFerroviario.chegadaTerminal2(this);

              while (getPartidaPrevista().isAfter(getCentral().getHoras())) {
                terminalFerroviario.esperarAtePartir();
              }

              if (getCentral().devoEsperar(this.getDestino()))
                terminalFerroviario.tolerancia(this);

              this.terminalFerroviario.partida(1, this);

            } else
              terminalFerroviario.esperarParaEstacionar();
```

```
                else
                    terminalFerroviario.esperaAteLivre();
                break;
            }
        }
    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder();

        str.append(super.toString()).append("|");

        return str.toString();

    }
}
```

Classe Transporte

```
public class Transporte {

    private int numero;
    private String destino;
    private int terminal;
    private LocalTime chegadaPrevista;
    private LocalTime partidaPrevista;
    private LocalTime chegada;
    private LocalTime partida;
    private int capacidade;
    private int lugaresOcupados;
    private Central central;
    private int chegadaDiaria;
    private long atrasoChegadaMinutos;
    private long atrasoPartidaMinutos;
    private LocalTime partidaSemTolerancia;
    private int passageirosDentroDoHorario;
    private int passageirosForaDoHorario;

    public Transporte(Central central) {
        this.central = central;
        this.chegadaDiaria = 0;
        this.atrasoChegadaMinutos = 0;
        this.atrasoPartidaMinutos = 0;
        this.partidaSemTolerancia = null;
        this.passageirosDentroDoHorario = 0;
        this.passageirosForaDoHorario = 0;
```

```
  }

  public Transporte(int numero, String destino, int terminal,
      LocalTime chegadaPrevista, LocalTime partidaPrevista, LocalTime horaActual, int
capacidade,
      int lugaresOcupados, Central central) {
    this.numero = numero;
    this.destino = destino;
    this.terminal = terminal;
    this.chegadaPrevista = chegadaPrevista;
    this.partidaPrevista = partidaPrevista;
    this.capacidade = capacidade;
    this.lugaresOcupados = lugaresOcupados;
    this.central = central;
    this.chegadaDiaria = 0;
    this.atrasoChegadaMinutos = 0;
    this.atrasoPartidaMinutos = 0;
    this.partidaSemTolerancia = null;
    this.passageirosDentroDoHorario = 0;
    this.passageirosForaDoHorario = 0;
  }

  public int getNumero() {
    return numero;
  }

  public void setNumero(int numero) {
    this.numero = numero;
  }

  public String getDestino() {
    return destino;
  }

  public void setDestino(String destino) {
    this.destino = destino;
  }

  public int getTerminal() {
    return terminal;
  }

  public void setTerminal(int terminal) {
    this.terminal = terminal;
  }

  public int getCapacidade() {
```

```
    return capacidade;
}

public void setCapacidade(int capacidade) {
    this.capacidade = capacidade;
}

public int getLugaresOcupados() {
    return lugaresOcupados;
}

public synchronized void setLugaresOcupados(int lugaresOcupados) {
    this.lugaresOcupados = lugaresOcupados;
}

public Central getCentral() {
    return central;
}

public void setCentral(Central central) {
    this.central = central;
}

public LocalTime getChegadaPrevista() {
    return chegadaPrevista;
}

public void setChegadaPrevista(LocalTime chegadaPrevista) {
    this.chegadaPrevista = chegadaPrevista;
}

public LocalTime getPartidaPrevista() {
    return partidaPrevista;
}

public void setPartidaPrevista(LocalTime partidaPrevista) {
    this.partidaPrevista = partidaPrevista;
}

public LocalTime getChegada() {
    return chegada;
}

public synchronized void setChegada(LocalTime chegada) {
    this.chegada = chegada;
}
```

```
public LocalTime getPartida() {
   return partida;
}

public synchronized void setPartida(LocalTime partida) {
   this.partida = partida;
}

public int getChegadaDiaria() {
   return chegadaDiaria;
}

public synchronized void setChegadaDiaria(int chegadaDiaria) {
   this.chegadaDiaria = chegadaDiaria;
}

public long getAtrasoChegadaMinutos() {
   return atrasoChegadaMinutos;
}

public synchronized void setAtrasoChegadaMinutos(long atrasoChegadaMinutos) {
   this.atrasoChegadaMinutos = atrasoChegadaMinutos;
}

public long getAtrasoPartidaMinutos() {
   return atrasoPartidaMinutos;
}

public synchronized void setAtrasoPartidaMinutos(long atrasoPartidaMinutos) {
   this.atrasoPartidaMinutos = atrasoPartidaMinutos;
}

public LocalTime getPartidaSemTolerancia() {
   return partidaSemTolerancia;
}

public synchronized void setPartidaSemTolerancia(LocalTime partidaSemTolerancia) {
   this.partidaSemTolerancia = partidaSemTolerancia;
}

public int getPassageirosDentroDoHorario() {
   return passageirosDentroDoHorario;
}

public synchronized void setPassageirosDentroDoHorario(int passageirosDentroDoHorario) {
   this.passageirosDentroDoHorario = passageirosDentroDoHorario;
}
```

```
    public int getPassageirosForaDoHorario() {
        return passageirosForaDoHorario;
    }

    public synchronized void setPassageirosForaDoHorario(int passageirosForaDoHorario) {
        this.passageirosForaDoHorario = passageirosForaDoHorario;
    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder();

        str.append("Numero: ").append(getNumero()).append("|");
        str.append("Destino: ").append(getDestino()).append("|");
        str.append("Chegada: ").append(getChegadaPrevista()).append("|");
        str.append("Partida: ").append(getPartidaPrevista()).append("|");
        str.append("Porta: ").append(getTerminal()).append("|");
        str.append("Capacidade: ").append(getCapacidade());

        return str.toString();

    }
}
```

Classe Aeroporto

```
public class Aeroporto {

    private static final int MAX_PISTAS = 3;

    private ArrayList<Aviao> avioes;
    private int[] avioesAterrados;
    private Central central;

    public Aeroporto(Central central) {
        this.avioes = new ArrayList<>();
        this.avioesAterrados = new int[MAX_PISTAS];
        this.central = central;

    }

    public ArrayList<Aviao> getAvioes() {
        return avioes;
    }

    public void setAvioes(ArrayList<Aviao> avioes) {
        this.avioes = avioes;
    }
```

```
  public Central getCentral() {
    return central;
  }

  public void setCentral(Central central) {
    this.central = central;
  }

  public int[] getAvioesAterrados() {
    return avioesAterrados;
  }

  public void setAvioesAterrados(int[] avioesAterrados) {
    this.avioesAterrados = avioesAterrados;
  }

  /**
   * Método para definir a hora de chegada dos grupos
   *
   * @param grupos grupos que chegaram
   */
  public synchronized void horaDeChegadaGrupos(ArrayList<Grupo> grupos) {
    for (Grupo g : grupos) {
      g.setHoraChegada(getCentral().getHoras());
    }

  }

  /**
   * Método boolean se indica se na plataforma i existe um aviao aterrado
   *
   * @param i numero da plataforma
   * @return true se a pista estiver livre = 0 false se = 1
   */
  public synchronized boolean pistaLivre(int i) {
    return this.avioesAterrados[i] == 0;
  }

  /**
   * Método para aterrar um aviao na plataforma 1
   *
   * @param aviao aviao que vai aterrar
   */
  public synchronized void aterra1(Aviao aviao) {
    System.out.println(Thread.currentThread().getName() + " aterrou às " + central.getHoras()
+ " na pista " + aviao.getTerminal());
```

```
        this.avioesAterrados[0]++;
        horaDeChegadaGrupos(aviao.getGrupos());

    }

    /**
     * Método para aterrar um aviao na plataforma 2
     *
     * @param aviao aviao que vai aterrar
     */
    public synchronized void aterra2(Aviao aviao) {
        System.out.println(Thread.currentThread().getName() + " aterrou às " + central.getHoras()
+ " na pista " + aviao.getTerminal());
        this.avioesAterrados[1]++;
        horaDeChegadaGrupos(aviao.getGrupos());

    }

    /**
     * Método para aterrar um aviao na plataforma 3
     *
     * @param aviao aviao que vai aterrar
     */
    public synchronized void aterra3(Aviao aviao) {
        System.out.println(Thread.currentThread().getName() + " aterrou às " + central.getHoras()
+ " na pista " + aviao.getTerminal());
        this.avioesAterrados[2]++;
        horaDeChegadaGrupos(aviao.getGrupos());

    }

    /**
     * Método para que o avião possa partir do aeroporto
     *
     * @param i plataforma de onde o aviao vai partir
     * @param aviao aviao a partir
     */
    public synchronized void partida(int i, Aviao aviao) {
        System.out.println(Thread.currentThread().getName() + " partiu às " + central.getHoras() +
" da pista " + (i + 1));
        this.avioesAterrados[i]--;
        aviao.setDescolou(true);
        notifyAll();
    }

    /**
     * Método para limpar o aviao durante 30min
```

```
     */
    public synchronized void LimparAviao() {
        System.out.println("Todos os passageiros já sairam do " +
Thread.currentThread().getName());
        System.out.println("Em limpeza");

        try {
            wait(300);
        } catch (InterruptedException ex) {
            Logger.getLogger(Aeroporto.class.getName()).log(Level.SEVERE, null, ex);
        }

    }

    /**
     * Método que vai confirmar se todos os grupos existentes num avião já
     * embarcaram ou nos comboios ou nos autocarros
     *
     * @param grupo grupo
     * @param aviao aviao a desembarcar
     */
    public synchronized void gruposEmbarcaram(ArrayList<Grupo> grupo, Aviao aviao) {
        for (Grupo g : grupo) {
            if (g.isEmbarcou() == true) {
                aviao.setTodosEmbarcaram(true);
            } else {
                aviao.setTodosEmbarcaram(false);
                break;
            }
        }

    }

    /**
     * Método que devolve quantos grupos ainda precisam de embarcar
     *
     * @param grupos grupos que precisam de embarcar
     * @return quantos faltam
     */
    public synchronized int gruposFaltam(ArrayList<Grupo> grupos) {
        int faltam = 0;
        for (Grupo g : grupos) {
            if (g.isEmbarcou() == false) {
                faltam++;
            }

        }
```

```
    return faltam;
}

/**
 * Método boolean que retorna true caso o aviao já tenha aterrado
 *
 * @param aviao que está a tentar aterrar
 * @return true se já aterrou = 1 false se não aterrou = 0
 */
public synchronized boolean jaAterrou(Aviao aviao) {
    return aviao.getAterragemDiaria() == 1;

}

/**
 * Método caso a pista de aterragem não esteja livre mete a thread a dormir
 */
public synchronized void esperaPorPistaLivre() {

    try {
        wait();
    } catch (InterruptedException ex) {
        Logger.getLogger(Aeroporto.class.getName()).log(Level.SEVERE, null, ex);
    }

}

/**
 * Método caso haja grupos no aviao e não haja autocarros ou comboios
 * disponiveis o aviao dorme
 */
public void esperaPorComboiosOuAutocarros() {
    try {
        Thread.sleep(100);
    } catch (InterruptedException ex) {
        Logger.getLogger(Aeroporto.class.getName()).log(Level.SEVERE, null, ex);
    }

}

/**
 * Método que diz ao aviao para dormir se ainda não pode aterrar
 */
public void esperaPorHoraDeAterrar() {

    try {
        Thread.sleep(100);
```

```
    } catch (InterruptedException ex) {
      Logger.getLogger(Aeroporto.class.getName()).log(Level.SEVERE, null, ex);
    }

  }

  /**
   * Método que cria os aviões a partir de um ficheiro
   *
   * @throws IOException
   */
  public void criarAvioes() throws IOException {
    this.avioes.add(new Aviao(this, central));
    File readingFile = new File("voos.txt");
    try {
      FileReader fileReader = new FileReader(readingFile);
      BufferedReader bufferedReader = new BufferedReader(fileReader);

      int i = 0;//váriavel de criação
      int j = 0;//numero de linha

      String line = "";
      while (line != null) {

        switch (j) {
          case 0:
            break;
          case 1://define o numero do aviao
            this.avioes.get(i).setNumero(Integer.parseInt(line));
            break;
          case 2://define o destino
            this.avioes.get(i).setOrigem(line);
            break;
          case 3://define a hora de chegada
            String str5[] = line.split(":");
            this.avioes.get(i).setChegadaPrevista(LocalTime.of(Integer.parseInt(str5[0]),
Integer.parseInt(str5[1])));
            break;
          case 4://defini a porta de desenbarque
            this.avioes.get(i).setTerminal(Integer.parseInt(line));
            break;
          default://define um grupo de passageiros
            String str[] = line.split(";");
            this.avioes.get(i).getGrupos().add(new Grupo(Integer.parseInt(str[0]), str[1],
str[2], this.avioes.get(i).getNumero()));
            break;
        }
```

```
            line = bufferedReader.readLine();//lê uma linha
            j++;//cada vez que lê uma linha incrementa

            //se for igual quer dizer que não existe mais informação deste aviao
            if ("".equals(line)) {
               this.avioes.add(new Aviao(this, this.central)); //cria-se um novo avião
               i++;
               j = 0;//volta a zera porque vamos começar a ler outro avião
            }

         }
      } catch (IOException e) {
         System.out.println(e.getMessage());

      }

   }

   /**
    * Método para imprimir os horarios dos avioes
    */
   public void horarioAvioes() {
      System.out.println("***Avioes***");
      for (Aviao avioe : this.avioes) {
         System.out.println(avioe.toString());
      }
   }

}
```

Classe Central

```
public class Central {

   private Aeroporto aeroporto;
   private TerminalFerroviario terminalFerroviario;
   private TerminalRodoviario terminalRodoviario;
   private Hora horas;

   public Central() {
      this.aeroporto = new Aeroporto(this);
      this.terminalFerroviario = new TerminalFerroviario(this);
      this.terminalRodoviario = new TerminalRodoviario(this);

   }

   public Aeroporto getAeroporto() {
```

```
    return aeroporto;
}

public void setAeroporto(Aeroporto aeroporto) {
    this.aeroporto = aeroporto;
}

public TerminalFerroviario getTerminalFerroviario() {
    return terminalFerroviario;
}

public void setTerminalFerroviario(TerminalFerroviario terminalFerroviario) {
    this.terminalFerroviario = terminalFerroviario;
}

public TerminalRodoviario getTerminalRodoviario() {
    return terminalRodoviario;
}

public void setTerminalRodoviario(TerminalRodoviario terminalRodoviario) {
    this.terminalRodoviario = terminalRodoviario;
}

public LocalTime getHoras() {
    return horas.getTime();
}

public boolean hasGrupos() {
    return false;
}

/**
 * Método para começar a thread de horas
 *
 * @param tempo horas
 */
public void startHoras(LocalTime tempo) {
    this.horas = new Hora(tempo, this);
    horas.start();
}

/**
 * Método boleano que return true caso haja um grupo num aviao por aterrar
 * que precise de apnhar o transporte
 *
 * @param destino para onde vai o grupo
 * @return true se houver passageiros false se não
```

```
 */
public synchronized boolean devoEsperar(String destino) {
   for (Aviao a : aeroporto.getAvioes()) {
      if (a.getAterragemDiaria() != 1)
         for (Grupo g : a.getGrupos()) {
            if (g.getDestino().equals(destino))
               return true;

         }
   }
   return false;
}

/**
 * Método para criar os transportes
 */
public void Criar() {

   try {
      this.aeroporto.criarAvioes();
      this.terminalFerroviario.criarComboios();
      this.terminalRodoviario.criarAutocarros();
   } catch (IOException ex) {
      Logger.getLogger(Central.class.getName()).log(Level.SEVERE, null, ex);
   }

}

/**
 * Método para imprimir os horarios de todos os transportes criados
 */
public void Horarios() {
   System.out.println("HORAS:" + horas.getTime());
   this.aeroporto.horarioAvioes();
   this.terminalFerroviario.horarioComboios();
   this.terminalRodoviario.horarioAutocarros();
}

/**
 * Método para saber quantos avioes descolaram
 *
 * @return numero de avioes que descolaram
 */
public int descolaram() {
   int descolaram = 0;
   for (Aviao a : aeroporto.getAvioes()) {
      if (a.isDescolou())
```

```
            descolaram++;
        }

        return descolaram;
    }

    /**
     * Método para saber quantos avioes descolaram
     *
     * @return numero de avioes que não descolaram
     */
    public int naoDescolaram() {
        int naoDescolaram = 0;
        for (Aviao a : aeroporto.getAvioes()) {
            if (!a.isDescolou())
                naoDescolaram++;

        }
        return naoDescolaram;
    }

    /**
     * Método para embarcar os passageiros nos comboios
     *
     * @param grupo a embarcar
     * @param aviao a desembarcar
     */
    public synchronized void embarcarComboio(Grupo grupo, Aviao aviao) {
        for (Comboio c : terminalFerroviario.getComboioParado()) {
            if (c != null)
                if (grupo.isEmbarcou() != true && c.getCapacidade() >= c.getLugaresOcupados()
                        && (c.getLugaresOcupados() + grupo.getNumeroPessoas() <= c.getCapacidade()))
                    if (grupo.getDestino().equals(c.getDestino())) {
                        c.setLugaresOcupados(c.getLugaresOcupados() + grupo.getNumeroPessoas());
                        c.getGruposComboio().add(grupo);
                        grupo.setEmbarcou(true);
                        grupo.setHoraEntrada(getHoras());
                        grupo.setTempoEspera(ChronoUnit.MINUTES.between(grupo.getHoraChegada(),
grupo.getHoraEntrada()));

                        System.out.println("O grupo de " + grupo.getDestino()
                                + " que veio no aviao " + aviao.getNumero()
                                + " entrou no comboio " + c.getNumero()
                                + " às " + grupo.getHoraEntrada());
                        System.out.println("Comboio " + c.getNumero()
                                + " com destino a " + c.getDestino() + " tem "
                                + c.getLugaresOcupados() + " pessoas");
```

```
            }
        }
    }

    /**
     * Método para embarcar os passageiros nos autocarros
     *
     * @param grupo a embarcar
     * @param aviao que está a desembarcar
     */
    public synchronized void embarcarAutocarro(Grupo grupo, Aviao aviao) {
        for (Autocarro a : terminalRodoviario.getAutocarroParado()) {
            if (a != null)
                if (grupo.isEmbarcou() != true && a.getCapacidade() >= a.getLugaresOcupados()
                        && (a.getLugaresOcupados() + grupo.getNumeroPessoas() <= a.getCapacidade()))
                    if (grupo.getDestino().equals(a.getDestino())) {
                        a.setLugaresOcupados(a.getLugaresOcupados() + grupo.getNumeroPessoas());
                        a.getGruposAutocarro().add(grupo);
                        grupo.setEmbarcou(true);
                        grupo.setHoraEntrada(getHoras());
                        grupo.setTempoEspera(ChronoUnit.MINUTES.between(grupo.getHoraChegada(),
grupo.getHoraEntrada()));

                        System.out.println("O grupo de " + grupo.getDestino()
                                + " que veio no aviao " + aviao.getNumero()
                                + " entrou no autocarro " + a.getNumero()
                                + " às " + grupo.getHoraEntrada());
                        System.out.println("Autocarro " + a.getNumero()
                                + " tem " + a.getLugaresOcupados() + " pessoas");

                    }
        }
    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder();

        str.append("****AVIOES QUE NÃO DESEMBARCARAM TODOS OS PASSAGEIROS = ")
                .append(naoDescolaram()).append("****\n");
        for (Aviao a : aeroporto.getAvioes()) {

            if (!a.isDescolou() && a.getChegada() != null) {

                str.append("\nAviao ").append(a.getNumero()).append(" aterrou às ")
                        .append(a.getChegada()).append("\n");
```

```
        str.append("**********Informação dos grupos********** \n");
        for (Grupo g : a.getGrupos()) {
          if (g.isEmbarcou())
            str.append("Grupo de ").append(g.getDestino())
                .append(" embarcou às ").append(g.getHoraEntrada())
                .append("\n");
          else
            str.append("Grupo de ").append(g.getDestino())
                .append(" não embarcou \n");
        }
        str.append("****************************************** \n");

      } else if (!a.isDescolou() && a.getChegada() == null) {

        str.append("\nAviao ").append(a.getNumero()).append(" não aterrou \n");
        str.append("**********Informação dos grupos********** \n");
        for (Grupo g : a.getGrupos()) {
          str.append("Grupo de ").append(g.getDestino())
              .append(" não embarcou \n");
        }
        str.append("******************************************\n");
      }

    }
    str.append("\n****AVIOES QUE DSEMBARCARAM TODOS OS PASSAGEIROS = ")
        .append(descolaram()).append("**** \n");
    for (Aviao a : aeroporto.getAvioes()) {
      if (a.isDescolou()) {

        str.append("\nAviao ").append(a.getNumero()).append(" aterrou às ")
            .append(a.getChegada()).append(" e partiu às ")
            .append(a.getPartida()).append("\n");
        str.append("**********Informação dos grupos**********").append("\n");
        for (Grupo g : a.getGrupos()) {

          if (g.isEmbarcou())
            str.append("Grupo de ").append(g.getDestino())
                .append(" embarcou às ").append(g.getHoraEntrada()).append("\n");

        }
        str.append("****************************************** \n");
      }
    }
    str.append("\n");
    str.append("******INFORMAÇÃO DOS COMBOIOS****** \n");
    for (Comboio comboio : terminalFerroviario.getComboios()) {
      if (comboio.getAtrasoChegadaMinutos() > 0) {
```

```
            str.append("\nComboio ").append(comboio.getNumero())
                .append(" com destino a ").append(comboio.getDestino()).append(" chegou às ")
                .append(comboio.getChegada()).append(" com um atraso de ")
                .append(comboio.getAtrasoChegadaMinutos())
                .append(" minutos partiu às ").append(comboio.getPartida())
                .append(" com um atraso de ").append(comboio.getAtrasoPartidaMinutos())
                .append(" minutos e com ").append(comboio.getLugaresOcupados())
                .append(" passageiros \n");

            if (comboio.getLugaresOcupados() != 0) {
              str.append("Grupos que embarcaram: \n");
              for (Grupo grupo : comboio.getGruposComboio()) {
                str.append("Grupo de ").append(grupo.getNumeroPessoas())
                    .append(" pessoas que chegou no avião ").append(grupo.getNumeroAviao())
                    .append(" entrou no comboio às ").append(grupo.getHoraEntrada())
                    .append(" e esperou ").append(grupo.getTempoEspera())
                    .append(" minutos \n");
                if (grupo.getHoraEntrada() == grupo.getHoraChegada())

comboio.setPassageirosDentroDoHorario(comboio.getPassageirosDentroDoHorario() +
grupo.getNumeroPessoas());
                else

comboio.setPassageirosForaDoHorario(comboio.getPassageirosForaDoHorario() +
grupo.getNumeroPessoas());

              }
              str.append("Numero de passageiros dentro do horario:
").append(comboio.getPassageirosDentroDoHorario()).append("\n");
              str.append("Numero de passageiros fora do horario:
").append(comboio.getPassageirosForaDoHorario()).append("\n");

            } else
              str.append("Não teve embarcações \n");

          } else {
            str.append("\nComboio ").append(comboio.getNumero())
                .append(" com destino a ").append(comboio.getDestino())
                .append(" chegou às ").append(comboio.getChegada())
                .append(" a horas partiu às ").append(comboio.getPartida())
                .append(" com um atraso de ").append(comboio.getAtrasoPartidaMinutos())
                .append(" minutos e com ").append(comboio.getLugaresOcupados())
                .append(" passageiros \n");
            if (comboio.getLugaresOcupados() != 0) {
              str.append("Grupos que embarcaram:\n");
              for (Grupo grupo : comboio.getGruposComboio()) {
```

```
                str.append("Grupo de ").append(grupo.getNumeroPessoas())
                        .append(" pessoas que chegou no avião ").append(grupo.getNumeroAviao())
                        .append(" entrou no comboio às ").append(grupo.getHoraEntrada())
                        .append(" e esperou ").append(grupo.getTempoEspera())
                        .append(" minutos \n");
                if (grupo.getHoraEntrada() == grupo.getHoraChegada())

comboio.setPassageirosDentroDoHorario(comboio.getPassageirosDentroDoHorario()          +
grupo.getNumeroPessoas());
                else

comboio.setPassageirosForaDoHorario(comboio.getPassageirosForaDoHorario()              +
grupo.getNumeroPessoas());

            }
            str.append("Numero de passageiros dentro do horario:
").append(comboio.getPassageirosDentroDoHorario()).append("\n");
            str.append("Numero de passageiros fora do horario:
").append(comboio.getPassageirosForaDoHorario()).append("\n");
          } else
            str.append("Não teve embarcações \n");

        }

    }
    str.append("****************************************** \n");
    str.append("\n******INFORMAÇÃO DOS Autocarros****** \n");
    for (Autocarro autocarro : terminalRodoviario.getAutocarros()) {
      if (autocarro.getAtrasoChegadaMinutos() > 0) {

        str.append("\nAutocarro ").append(autocarro.getNumero())
            .append(" com destino a ").append(autocarro.getDestino()).append(" chegou às ")
            .append(autocarro.getChegada()).append(" com um atraso de ")
            .append(autocarro.getAtrasoChegadaMinutos())
            .append(" minutos partiu às ").append(autocarro.getPartida())
            .append(" com um atraso de ").append(autocarro.getAtrasoPartidaMinutos())
            .append(" minutos e com ").append(autocarro.getLugaresOcupados())
            .append(" passageiros \n");

        if (autocarro.getLugaresOcupados() != 0) {
          str.append("Grupos que embarcaram: \n");
          for (Grupo grupo : autocarro.getGruposAutocarro()) {
            str.append("Grupo de ").append(grupo.getNumeroPessoas())
                .append(" pessoas que chegou no avião ").append(grupo.getNumeroAviao())
                .append(" entrou no comboio às ").append(grupo.getHoraEntrada())
                .append(" e esperou ").append(grupo.getTempoEspera())
                .append(" minutos \n");
```

```
                if (grupo.getHoraEntrada() == grupo.getHoraChegada())

autocarro.setPassageirosDentroDoHorario(autocarro.getPassageirosDentroDoHorario()      +
grupo.getNumeroPessoas());
                else

autocarro.setPassageirosForaDoHorario(autocarro.getPassageirosForaDoHorario()          +
grupo.getNumeroPessoas());

            }
            str.append("Numero de passageiros dentro do horario:
").append(autocarro.getPassageirosDentroDoHorario()).append("\n");
            str.append("Numero de passageiros fora do horario:
").append(autocarro.getPassageirosForaDoHorario()).append("\n");

          } else
            str.append("Não teve embarcações \n");

        } else {
          str.append("\nAutocarro ").append(autocarro.getNumero())
              .append(" com destino a ").append(autocarro.getDestino())
              .append(" chegou às ").append(autocarro.getChegada())
              .append(" a horas partiu às ").append(autocarro.getPartida())
              .append(" com um atraso de ").append(autocarro.getAtrasoPartidaMinutos())
              .append(" minutos e com ").append(autocarro.getLugaresOcupados())
              .append(" passageiros \n");
          if (autocarro.getLugaresOcupados() != 0) {
            str.append("Grupos que embarcaram:\n");
            for (Grupo grupo : autocarro.getGruposAutocarro()) {
              str.append("Grupo de ").append(grupo.getNumeroPessoas())
                  .append(" pessoas que chegou no avião ").append(grupo.getNumeroAviao())
                  .append(" entrou no comboio às ").append(grupo.getHoraEntrada())
                  .append(" e esperou ").append(grupo.getTempoEspera())
                  .append(" minutos \n");

                if (grupo.getHoraEntrada() == grupo.getHoraChegada())

autocarro.setPassageirosDentroDoHorario(autocarro.getPassageirosDentroDoHorario()      +
grupo.getNumeroPessoas());
                else

autocarro.setPassageirosForaDoHorario(autocarro.getPassageirosForaDoHorario()          +
grupo.getNumeroPessoas());

            }
            str.append("Numero de passageiros dentro do horario:
").append(autocarro.getPassageirosDentroDoHorario()).append("\n");
```

```
            str.append("Numero de passageiros fora do horario: ").append(autocarro.getPassageirosForaDoHorario()).append("\n");

            } else
                str.append("Não teve embarcações \n");

        }

    }
    str.append("****************************************** \n");
    return str.toString();
  }
}
```

Classe TerminalFerroviario

```
public class TerminalFerroviario {

  private static final int PLATAFORMAS = 2;

  private ArrayList<Comboio> comboios;
  private int[] plataformaOcupada;
  private Comboio[] comboioParado;
  private Central central;

  public TerminalFerroviario(Central central) {
    this.comboios = new ArrayList<>();
    this.plataformaOcupada = new int[PLATAFORMAS];
    this.central = central;
    this.comboioParado = new Comboio[PLATAFORMAS];

  }

  public ArrayList<Comboio> getComboios() {
    return comboios;
  }

  public void setComboios(ArrayList<Comboio> comboios) {
    this.comboios = comboios;
  }

  public int[] getPlataformaOcupada() {
    return plataformaOcupada;
  }

  public void setPlataformaOcupada(int[] plataformaOcupada) {
    this.plataformaOcupada = plataformaOcupada;
```

```
}

public Central getCentral() {
   return central;
}

public void setCentral(Central central) {
   this.central = central;
}

public Comboio[] getComboioParado() {
   return comboioParado;
}

public void setComboioParado(Comboio[] comboioParado) {
   this.comboioParado = comboioParado;
}

/**
 * Método para criar comboios a partir de um ficheiro txt
 *
 * @throws IOException exceção
 */
public void criarComboios() throws IOException {

   this.comboios.add(new Comboio(this.central, this));

   File readingFile = new File("comboios.txt");
   try {
      FileReader fileReader = new FileReader(readingFile);
      BufferedReader bufferedReader = new BufferedReader(fileReader);

      int i = 0;
      int j = 0;

      String line = "";
      while (line != null) {
         switch (j) {
            case 1:
               this.comboios.get(i).setNumero(Integer.parseInt(line));
               break;
            case 2:
               this.comboios.get(i).setDestino(line);
               break;
            case 3:
               String str[] = line.split(":");
```

```
                this.comboios.get(i).setChegadaPrevista(LocalTime.of(Integer.parseInt(str[0]),
Integer.parseInt(str[1])));
                break;
            case 4:
                String str1[] = line.split(":");
                this.comboios.get(i).setPartidaPrevista(LocalTime.of(Integer.parseInt(str1[0]),
Integer.parseInt(str1[1])));
                break;
            case 5:
                this.comboios.get(i).setTerminal(Integer.parseInt(line));
                break;
            case 6:
                this.comboios.get(i).setCapacidade(Integer.parseInt(line));
                break;

          }

          line = bufferedReader.readLine();
          j++;

          if ("".equals(line)) {

            this.comboios.add(new Comboio(this.central, this));
            i++;
            j = 0;

          }

        }
      } catch (IOException e) {
        System.out.println(e.getMessage());
      }

    }

    /**
     * Método para que caso uma thread queira estacionar se a plataforma estiver
     * ocupada fica à espera
     */
    public synchronized void esperaAteLivre() {
      try {
        wait();
      } catch (InterruptedException ex) {
        Logger.getLogger(TerminalFerroviario.class.getName()).log(Level.SEVERE, null, ex);
      }

    }
```

```
  /**
   * Método para adormecer a thread caso ainda exista uma aviao com pasageiros
   * a chegar
   *
   * @param comboio comboio que dorme
   */
  public void tolerancia(Comboio comboio) {
    System.out.println("Comboio " + comboio.getNumero() + "->Existe um avião a chegar com
passageiros"
        + " tempo de tolerancia: 10 minutos");
    try {
      Thread.sleep(101);
    } catch (InterruptedException ex) {
      Logger.getLogger(TerminalFerroviario.class.getName()).log(Level.SEVERE, null, ex);
    }

  }

  /**
   * Método para adormecer o comboio caso ainda não seja a hora de partida
   */
  public void esperarAtePartir() {
    try {
      Thread.sleep(100);
    } catch (InterruptedException ex) {
      Logger.getLogger(TerminalFerroviario.class.getName()).log(Level.SEVERE, null, ex);
    }

  }

  /**
   * Método para adormecer o comboio caso este ainda não posso estacionar
   */
  public void esperarParaEstacionar() {
    try {
      Thread.sleep(100);
    } catch (InterruptedException ex) {
      Logger.getLogger(TerminalFerroviario.class.getName()).log(Level.SEVERE, null, ex);
    }

  }

  /**
   * Método para imprimir o horário dos comboios
   */
  public void horarioComboios() {
```

```
        System.out.println("\n***Comboios***");
        this.comboios.stream().forEach((comboio) -> {
            System.out.println(comboio.toString());
        });
    }

    /**
     * Método para registar que um comboio chegou à 1 plataforma
     *
     * @param comboio comboio que chegou
     */
    public synchronized void chegadaTerminal1(Comboio comboio) {
        System.out.println(Thread.currentThread().getName() + " parou às " + central.getHoras());
        comboio.setChegada(getCentral().getHoras());
        this.plataformaOcupada[0]++;
        this.comboioParado[0] = comboio;
        notifyAll();
    }

    /**
     * Método para registar que um comboio chegou à 2 plataforma
     *
     * @param comboio comboio que chegou
     */
    public synchronized void chegadaTerminal2(Comboio comboio) {
        System.out.println(Thread.currentThread().getName() + " parou às " + central.getHoras());
        comboio.setChegada(getCentral().getHoras());
        this.plataformaOcupada[1]++;
        this.comboioParado[1] = comboio;
        notifyAll();
    }

    /**
     * Método que ser para registar a partida de um comboio
     *
     * @param i numero da plataforma
     * @param comboio que vai partir
     */
    public synchronized void partida(int i, Comboio comboio) {
        System.out.println(Thread.currentThread().getName() + " partiu às " + central.getHoras());
        comboio.setPartida(getCentral().getHoras());
        this.plataformaOcupada[i]--;
        this.comboioParado[i] = null;

comboio.setAtrasoChegadaMinutos(ChronoUnit.MINUTES.between(comboio.getChegadaPrevi
sta(), comboio.getChegada()));
```

```
comboio.setAtrasoPartidaMinutos(ChronoUnit.MINUTES.between(comboio.getPartidaPrevista
(), comboio.getPartida()));
      notifyAll();
   }

   /**
    * Método que devolve true caso a plataforma esteja livre
    *
    * @param i numero da plataforma
    * @return true se estiver livre ou seja igual a 0 ou false se estiver
    * ocupada
    */
   public boolean terminalLivre(int i) {
      return this.plataformaOcupada[i] == 0;

   }

}
```

Classe TerminalRodoviario

```
public class TerminalRodoviario {

   private static final int PLATAFORMAS = 3;

   private ArrayList<Autocarro> autocarros;
   private int[] plataformaOcupada;
   private Central central;
   private Autocarro[] autocarroParado;

   public TerminalRodoviario(Central central) {
      this.autocarros = new ArrayList<>();
      this.plataformaOcupada = new int[PLATAFORMAS];
      this.central = central;
      this.autocarroParado = new Autocarro[PLATAFORMAS];
   }

   public ArrayList<Autocarro> getAutocarros() {
      return autocarros;
   }

   public void setAutocarros(ArrayList<Autocarro> autocarros) {
      this.autocarros = autocarros;
   }

   public int[] getPlataformaOcupada() {
      return plataformaOcupada;
```

```
}

public void setPlataformaOcupada(int[] plataformaOcupada) {
   this.plataformaOcupada = plataformaOcupada;
}

public Central getCentral() {
   return central;
}

public void setCentral(Central central) {
   this.central = central;
}

public Autocarro[] getAutocarroParado() {
   return autocarroParado;
}

public void setAutocarroParado(Autocarro[] autocarroParados) {
   this.autocarroParado = autocarroParados;
}

/**
 * Método para a criação de autocarros a partir de um ficheiro txt
 *
 * @throws IOException exceçao
 */
public void criarAutocarros() throws IOException {

   this.autocarros.add(new Autocarro(this.central, this));

   File readingFile = new File("autocarros.txt");
   try {
      FileReader fileReader = new FileReader(readingFile);
      BufferedReader bufferedReader = new BufferedReader(fileReader);

      int i = 0;
      int j = 0;

      String line = "";
      while (line != null) {

         switch (j) {
            case 1:
               this.autocarros.get(i).setNumero(Integer.parseInt(line));
               break;
            case 2:
```

```
                this.autocarros.get(i).setDestino(line);
                break;
            case 3:
                String str[] = line.split(":");
                this.autocarros.get(i).setChegadaPrevista(LocalTime.of(Integer.parseInt(str[0]),
Integer.parseInt(str[1])));
                break;
            case 4:
                String str1[] = line.split(":");
                this.autocarros.get(i).setPartidaPrevista(LocalTime.of(Integer.parseInt(str1[0]),
Integer.parseInt(str1[1])));
                break;
            case 5:
                this.autocarros.get(i).setTerminal(Integer.parseInt(line));
                break;
            case 6:
                this.autocarros.get(i).setCapacidade(Integer.parseInt(line));
                break;

          }

          line = bufferedReader.readLine();
          j++;

          if ("".equals(line)) {
            this.autocarros.add(new Autocarro(this.central, this));
            i++;
            j = 0;

          }

        }
      } catch (IOException e) {
        System.out.println(e.getMessage());
      }

  }

  /**
   * Método para imprirmi os horários dos autocarros
   */
  public void horarioAutocarros() {
    System.out.println("\n***Autocarros***");
    for (Autocarro auto : this.autocarros) {
      System.out.println(auto.toString());
    }
  }
```

```
  /**
   * Método para fazer o autocarro esperar até a pista estar livre
   */
  public synchronized void esperarAteLivre() {
    try {
      wait();
    } catch (InterruptedException ex) {
      Logger.getLogger(TerminalRodoviario.class.getName()).log(Level.SEVERE, null, ex);
    }

  }

  /**
   * Método em poe o autocarro a dormir o tempo de tolerancia caso haja um
   * aviao a vir com passageiros para este autocarro
   *
   * @param autocarro autocarro que vai dormir
   */
  public void tolerancia(Autocarro autocarro) {
    System.out.println("Autocarro " + autocarro.getNumero() + "->Existe um avião a chegar
com passageiros"
        + " tempo de tolerancia: 20 minutos");
    try {
      Thread.sleep(201);
    } catch (InterruptedException ex) {
      Logger.getLogger(TerminalRodoviario.class.getName()).log(Level.SEVERE, null, ex);
    }

  }

  /**
   * Método para adormecer a thread enquanto esta não puder partir
   */
  public void esperarAtePartir() {
    try {
      Thread.sleep(100);
    } catch (InterruptedException ex) {
      Logger.getLogger(TerminalRodoviario.class.getName()).log(Level.SEVERE, null, ex);
    }

  }

  /**
   * Método que adormece a thread caso está ainda não possa estacionar
   */
  public void esperaParaEstacionar() {
```

```
    try {
        Thread.sleep(100);
    } catch (InterruptedException ex) {
        Logger.getLogger(TerminalRodoviario.class.getName()).log(Level.SEVERE, null, ex);
    }

}

/**
 * Método que defini quando o autocarro chega ao terminal 1
 *
 * @param autocarro autocarro que estaciouna
 */
public synchronized void chegadaTerminal1(Autocarro autocarro) {
    System.out.println(Thread.currentThread().getName() + " parou às " + central.getHoras());
    autocarro.setChegada(getCentral().getHoras());
    this.plataformaOcupada[0]++;
    this.autocarroParado[0] = autocarro;
    notifyAll();

}

/**
 * Método que defini quando o autocarro chega ao terminal 2
 *
 * @param autocarro autocarro que estaciouna
 */
public synchronized void chegadaTerminal2(Autocarro autocarro) {
    System.out.println(Thread.currentThread().getName() + " parou às " + central.getHoras());
    autocarro.setChegada(getCentral().getHoras());
    this.plataformaOcupada[1]++;
    this.autocarroParado[1] = autocarro;
    notifyAll();

}

/**
 * Método que defini quando o autocarro chega ao terminal 3
 *
 * @param autocarro autocarro que estaciouna
 */
public synchronized void chegadaTerminal3(Autocarro autocarro) {
    System.out.println(Thread.currentThread().getName() + " parou às " + central.getHoras());
    autocarro.setChegada(getCentral().getHoras());
    this.plataformaOcupada[2]++;
    this.autocarroParado[2] = autocarro;
    notifyAll();
```

```
    }

    /**
     * Método que define a partida do autocarro da plataforma i
     *
     * @param i plataforma
     * @param autocarro autocarro que vai partir
     */
    public synchronized void partida(int i, Autocarro autocarro) {
        System.out.println(Thread.currentThread().getName() + " partiu às " + central.getHoras());
        this.plataformaOcupada[i]--;
        this.autocarroParado[i] = null;
        autocarro.setPartida(getCentral().getHoras());

autocarro.setAtrasoChegadaMinutos(ChronoUnit.MINUTES.between(autocarro.getChegadaPr
evista(), autocarro.getChegada()));

autocarro.setAtrasoPartidaMinutos(ChronoUnit.MINUTES.between(autocarro.getPartidaPrevis
ta(), autocarro.getPartida()));
        notifyAll();
    }
    /**
     * Método boolean que devolve true caso a pltaforma não esteja ocupada
     * @param i plataforma
     * @return true se estiver livre = 0, false se estiver ocupada = 1
     */
    public boolean terminalLivre(int i) {
        return this.plataformaOcupada[i] == 0;

    }

}
```