

Felhasználói kézikönyv

Bevezetés

A Zedországi Államvasutak (ZÁV) a megnövekedett késések miatt drasztikus döntést hozott. A vonatok gyakran azért késtek, mert a pálya foglalt volt, ezért úgy döntöttek eltávolítják a biztosítóberendezéseket. Ez teljesen biztonságos, hiszen jól képzett és tehetséges forgalmisták, mint te fogják a vonatokat irányítani. Természetesen ez egy nagy felelősség. Bármilyen incidens történik az a te és csakis a te hibád nem a vasút társaságé.

Ez egy fiktív történet. Bármilyen párhuzam valós személyekhez vagy vasút társaságokhoz kizárólag a véletlen műve.

Pályák

A beépített pályákat a főmenüben lévő számozott gombokkal lehet elérni. Egyedi pályákat a „Play from file” gombbal lehet elérni. A játékban a bal oldali bejáratokon vonatok fognak érkezni. A bejáratok feletti pöttyök mutatják, hogy milyen színű és mikor érkezik a vonat. 3 pötty jelenti azt, hogy azonnal érkezik a következő.

Vonatok

5 színű vonat van. A piros, kék és zöld vonatok a legegyszerűbbek. Nekik meg kell állni a velük megegyező színű peronoknál. A fekete vonatok nem állnak meg egyik peronnál sem. A fehér vonatok bármely peronnál megállhatnak.

Játékmenet

A vonatokat a váltókra kattintással lehet más irányba terelni. Két vonat nem ütközhet össze, különben veszítesz. A vonatokat le lehet lassítani azzal, hogy nyomva tartod rajtuk az egered. Ha egy vonat elhagyja a pályát anélkül, hogy megállt volna (kivéve a fekete vonatokat) akkor is veszítesz.

Pálya szerkesztő

A pálya szerkesztő segítségével létre tudod hozni a saját pályádat. Importálhatsz meglévő pályát vagy kezdheted az alapoktól. A jobb oldali listából választhatsz pálya elemeket, amiket lerakhatsz. Ha valamit rossz helyre tettél akkor csak kattints rá és töröld ki a jobb oldalon lévő gombbal. Bátran kísérletezz!

```

classDiagram
    class Level {
        +String name
        +List<Train> trains
        +List<LevelComponent> components
        +int width
        +int height
        +verify() void
        +getComponentsInRect(int, int) List<LevelComponent>
        +getComponentAt(int, int) LevelComponent
        +List<LevelComponent> components
        +String name
        +List<Train> trains
        +int width
        +int height
    }
    class LevelReader {
        +readLevel() Level
    }
    class LevelWriter {
        +writeLevel(Level) void
    }
    class XmlLevelReader {
        +XmlLevelReader(String)
        +readLevel() Level
    }
    class XmlLevelWriter {
        +XmlLevelWriter(String)
        +writeLevel(Level) void
    }
    class LevelComponent {
        +LevelComponent(int, int, Coordinates)
        +height int
        +width int
        +verifyConstraints() void
        +accept(Visitor) void
        +width int
        +topLeftCorner Coordinates
        +height int
    }
    class LevelComponentFactory {
        +LevelComponentFactory()
        +getWidthFrom(Element) int
        +getCoordinatesFrom(Element) Coordinates
        +getColorFrom(Element) Colors
        +getHeightFrom(Element) int
        +getTrainsFrom(Element, List<TrainArrival>, Entrance) void
        +createLevelComponentFrom(Element) LevelComponent
    }
    class LevelComponentMarshaller {
        +LevelComponentMarshaller()
        +visit(Exit) void
        +visit(Junction) void
        +visit(Switch) void
        +createElement(String, LevelComponent) void
        +visit(Entrance) void
        +elementFrom(LevelComponent) Element
        +visit(Track) void
        +visit(Platform) void
    }
    LevelReader --> Level : readLevel()
    LevelWriter --> Level : writeLevel()
    XmlLevelReader --> Level : readLevel()
    XmlLevelWriter --> Level : writeLevel()
    LevelComponentFactory --> LevelComponent : createLevelComponentFrom()
    LevelComponentMarshaller --> LevelComponent : visit(), createElement(), elementFrom()
    
```

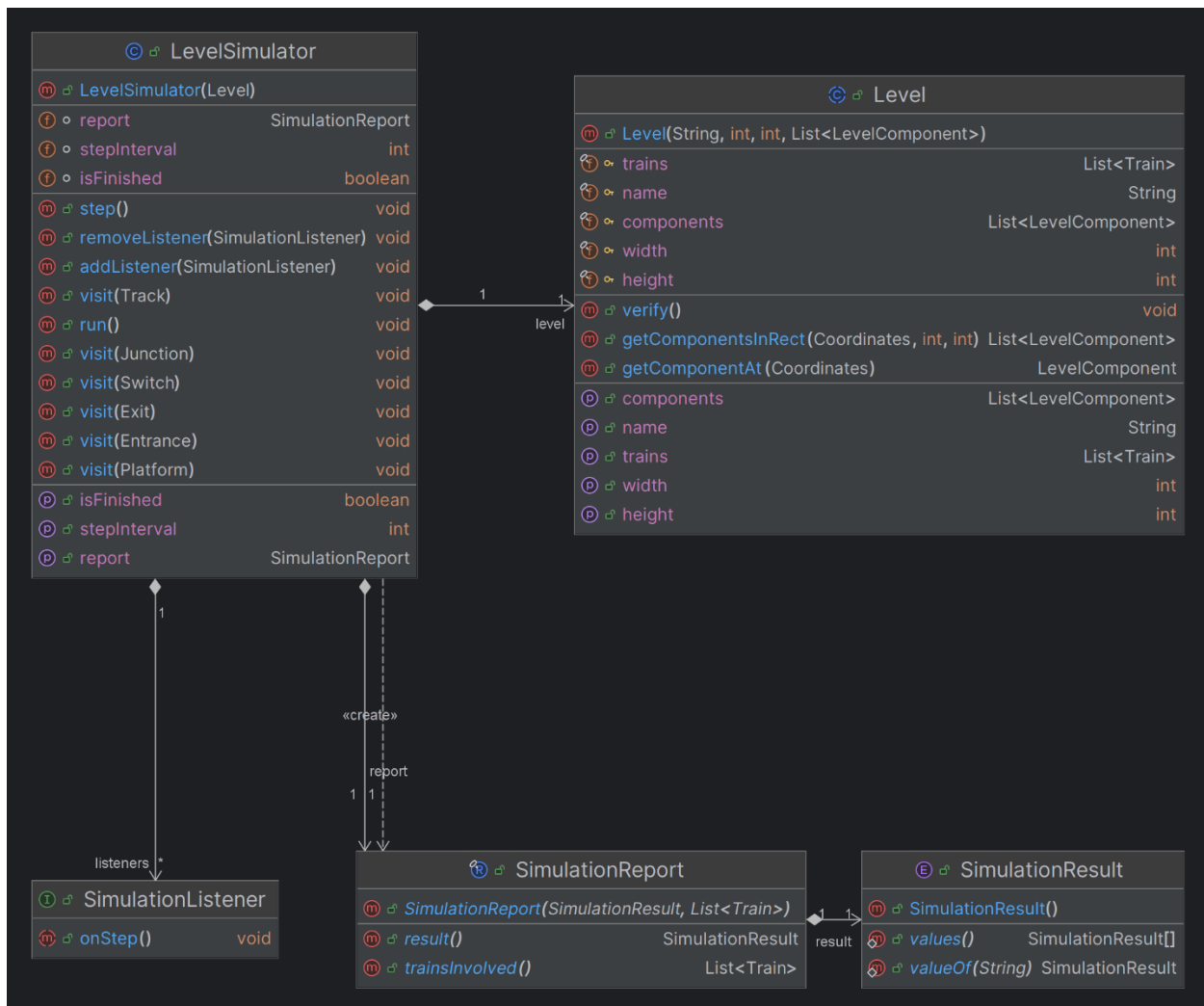
The diagram illustrates the relationships between the Level component and its associated classes. The Level class is the central entity, containing attributes like name, trains, components, width, and height, and methods for verification, retrieval, and writing. It is associated with LevelReader, LevelWriter, XmlLevelReader, and XmlLevelWriter. The LevelComponent class represents a component within a level, with attributes like height, width, and coordinates, and methods for verification, acceptance, and retrieval. The LevelComponentFactory class is responsible for creating LevelComponent objects. The LevelComponentMarshaller class handles the marshalling of LevelComponent objects into a format suitable for storage or transmission.

```

classDiagram
    class LevelComponent {
        +LevelComponent(int, int, Coordinates)
        +height int
        +width int
        +verifyConstraints(Level) void
        +accept(Visitor) void
        +width int
        +topLeftCorner Coordinates
        +height int
    }
    class Junction {
        +Junction(Coordinates, int)
        +segmentOnTrack TrainSegment
        +verifyConstraints(Level) void
        +navigate(TrainSegment) Navigable
        +accept(Visitor) void
        +segmentOnTrack TrainSegment
    }
    class Platform {
        +Platform(int, Coordinates, Colors)
        +color Colors
        +accept(Visitor) void
        +navigate(TrainSegment) Navigable
        +color Colors
    }
    class Track {
        +Track(int, Coordinates)
        +segmentsOnTrack Map<Integer, TrainSegment>
        +verifyConstraints(Level) void
        +accept(Visitor) void
        +navigate(TrainSegment) Navigable
        +segmentsOnTrack Map<Integer, TrainSegment>
    }
    class Level {
        +Level(String, int, int, List<LevelComponent>)
        +trains List<Train>
        +name String
        +components List<LevelComponent>
        +width int
        +height int
        +verify() void
        +getComponentsInRect(Coordinates, int, int) List<LevelComponent>
        +getComponentAt(Coordinates) LevelComponent
        +components List<LevelComponent>
        +name String
        +trains List<Train>
        +width int
        +height int
    }
    class Train {
        +Train(Navigable, int, Colors)
        +color Colors
        +reachedEnd boolean
        +segments List<TrainSegment>
        +move() void
        +stopFor(int) void
        +arrivedAtPlatform(Platform) void
        +reachedEnd boolean
        +completed boolean
        +segments List<TrainSegment>
        +color Colors
    }
    class TrainSegment {
        +TrainSegment(Train, Navigable, boolean)
        +isLeading boolean
        +train Train
        +move() boolean
        +isLeading boolean
        +train Train
    }
    class Entrance {
        +Entrance(Coordinates, List<TrainArrival>)
        +countdown int
        +waiting TrainSegment
        +trains List<TrainArrival>
        +navigate(TrainSegment) Navigable
        +step() void
        +accept(Visitor) void
        +verifyConstraints(Level) void
        +waiting TrainSegment
        +trains List<TrainArrival>
        +countdown int
    }
    class Exit {
        +Exit(Coordinates)
        +waiting TrainSegment
        +accept(Visitor) void
        +navigate(TrainSegment) Navigable
        +waiting TrainSegment
    }
    class DotComponent {
        +DotComponent()
        +DotComponent(Coordinates)
    }
    class TrainArrival {
        +TrainArrival()
        +TrainArrival(Train, int)
    }
    class Coordinates {
        +Coordinates(int, int)
        +y() int
        +x() int
        +transform(int, int) Coordinates
    }
    class StopAnywhereTrain {
        +StopAnywhereTrain(Navigable, int)
        +arrivedAtPlatform(Platform) void
        +completed boolean
    }
    class NonStopTrain {
        +NonStopTrain(Navigable, int)
        +arrivedAtPlatform(Platform) void
        +completed boolean
    }
    class StopAtColorTrain {
        +StopAtColorTrain(Navigable, int, Colors)
        +arrivedAtPlatform(Platform) void
        +completed boolean
    }
    class Visitor {
        +visit(Entrance) void
        +visit(Exit) void
        +visit(Platform) void
        +visit(Switch) void
        +visit(Junction) void
        +visit(Track) void
    }
    class OccupiedException {
        +OccupiedException(TrainSegment, Navigable)
        +occupiedBy TrainSegment
        +where Navigable
        +occupiedBy TrainSegment
        +where Navigable
    }
    class Switch {
        +Switch(Coordinates, int)
        +currentExit int
        +navigate(TrainSegment) Navigable
        +accept(Visitor) void
        +verifyConstraints(Level) void
        +segmentOnTrack TrainSegment
        +currentExit int
    }
    class ComponentConstraintException {
        +ComponentConstraintException(LevelComponent, String)
        +component LevelComponent
        +component LevelComponent
    }

    LevelComponent "1" -- "1" Junction
    Junction "1" -- "1" Platform
    Junction "1" -- "1" Track
    Platform "1" -- "1" Track
    Track "1" -- "1" Level
    Level "1" -- "1" Train
    Level "1" -- "1" TrainSegment
    Entrance "1" -- "1" TrainArrival
    Entrance "1" -- "1" Exit
    Entrance "1" -- "1" DotComponent
    Exit "1" -- "1" DotComponent
    DotComponent "1" -- "1" TrainArrival
    TrainArrival "1" -- "1" Coordinates
    Coordinates "1" -- "1" StopAnywhereTrain
    Coordinates "1" -- "1" NonStopTrain
    Coordinates "1" -- "1" StopAtColorTrain
    StopAnywhereTrain "1" -- "1" Train
    NonStopTrain "1" -- "1" Train
    StopAtColorTrain "1" -- "1" Train
    Visitor "1" -- "1" LevelComponent
    Visitor "1" -- "1" Junction
    Visitor "1" -- "1" Platform
    Visitor "1" -- "1" Track
    Visitor "1" -- "1" Level
    Visitor "1" -- "1" Entrance
    Visitor "1" -- "1" Exit
    Visitor "1" -- "1" DotComponent
    Visitor "1" -- "1" TrainArrival
    Visitor "1" -- "1" Coordinates
    OccupiedException "1" -- "1" TrainSegment
    OccupiedException "1" -- "1" Navigable
    Switch "1" -- "1" Junction
    Switch "1" -- "1" Entrance
    Switch "1" -- "1" Exit
    Switch "1" -- "1" DotComponent
    Switch "1" -- "1" TrainArrival
    Switch "1" -- "1" Coordinates
    ComponentConstraintException "1" -- "1" LevelComponent
    ComponentConstraintException "1" -- "1" LevelComponent
  
```

A szimuláció osztálydiagrammja



Modell

A pályát leíró modellt teljesen el akartam különíteni a szimulációtól és a megjelenítéstől. Gondot okozott, hogyan adjak egyedi megjelenést a Level-ben lévő heterogén kollekcióban lévő komponenseknek. Erre a Visitor mintát alkalmaztam. Sajnos ez igényel valamennyi kód ismétlést, mert minden osztálynak kell egy metódus a Visitor interfészben és az osztályban is, de még így is elegánsabb, mint instanceof-al megkülönböztetni az egyes komponenseket.

Szimuláció

A szimuláció felelőssége tetszőleges sebességgel léptetni a modellt és a modell állásából eldöntse véget ért-e a szimuláció és sikeres volt-e.

Ellenőrzés

A pályák beolvasása után egy ellenőrzés lépés következik, amiben az egyes komponensek követelményei le lesznek ellenőrizve, valamint a komponensek közötti kapcsolatok is létrejönnek. Jelenlegi implementációja nem túl hatékony egymásba ágyazott ciklus, aminek $O(n^2)$ komplexitása van, ahol n a komponensek száma. Mivel ez csak egyszer fut le betöltéskor elfogadhatónak tartom.

Nyelvi eszközök

Próbáltam a nyelv minél több elemét felhasználni fejlesztés közben, például szálkezelést és rekordokat a szimulációhoz, valamint számos helyen használtam lambda kifejezéseket.

Külső könyvtárak

- A program XML kezeléséhez a JDOM2 nevű könyvtárat használja.
- A program egységteszteléshez a JUnit5.8.1 és a testng könyvtárat használja.
- A függőségek pontosabban az „src.iml” fájlban vannak definiálva.

Dokumentálás

A metódusok dokumentálásához Javadoc kommenteket használtam a forráskódban.