Formalizing Domain Models in FOL

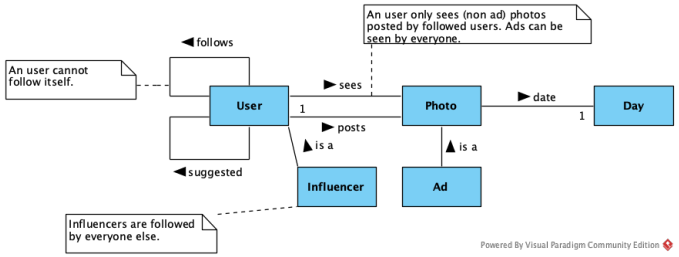
Maria João Frade

HASLab - INESC TEC
Departamento de Informática, Universidade do Minho

2023/24

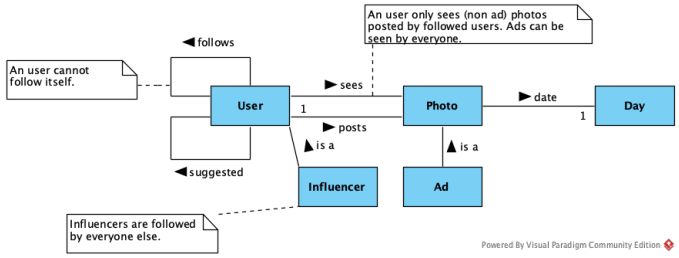
Domain models

- A **domain model** is a conceptual model of a system that describes the various entities involved in that system and their relationships.



- A domain model is a selective and structured representation of domain knowledge relevant to a given software development project.
- It is used in software development as a first step to realize a new domain and resolve ambiguities in requirements.

Domain models



A domain model is a diagram where

- entities are represented by boxes and
- relationships between entities by arcs annotated with
  - the name of the relationship/association,
  - the reading direction (indicated by an arrow) and
  - its multiplicity (annotated at the tip of the arcs).
- annotations with restrictions that are informally indicated in boxes.

Domain models

- There is no standard notation for domain models.
- We are going to use the notation used in the *Desenvolvimento de Sistemas de Software* course. So, we assume that:
  - all entities present in a diagram are disjoint, unless between two entities there is a relation/association *"is a"* which denotes a specialization/extension;
  - if there are multiple specializations for the same entity, those specializations are disjoint;
  - the relation/association *"is a"* may be a subset or a membership relation. If it is a membership relation, the entity that "belongs to" another will be modeled as a constant.

Formalizing a domain model

Our goal is to formalize domain models in many-sorted (typed) first-order logic with equality.

We start by establish the logical language that we are going to use, i.e., the vocabulary of the language. We will need

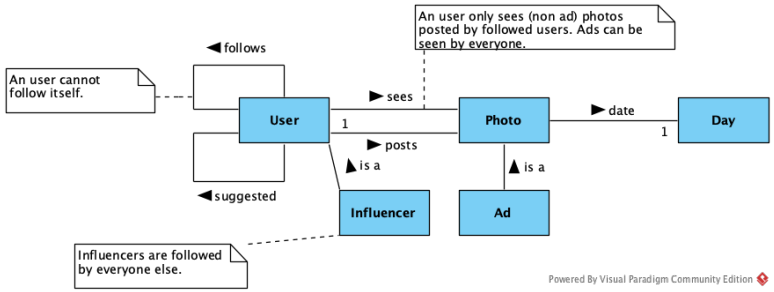
- to identify the types into which the universe of discourse is fragmented;
- a predicate for each subset relation
- a predicate for each association, except specializations (which will be codified by formulas that establish the kind of specialization);
- a constant for each entity belonging to an "enumerated type".

Formalizing a domain model

Next we write the set of formulas that describe the system. These formulas are of a different nature:

- codification of specialization relationships;
- disjunction of specialization entities;
- typing associations that involve subsets;
- multiplicity restrictions on associations;
- restrictions annotated informally.

Example: Instagram



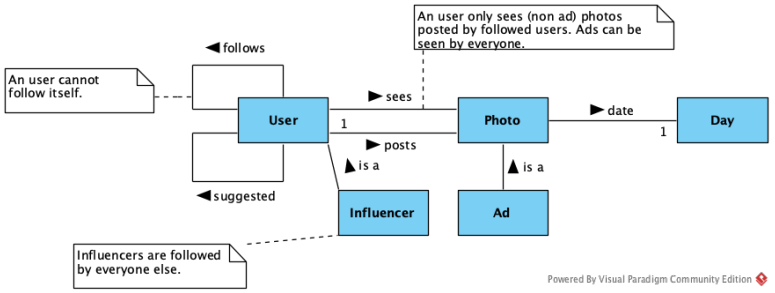
Types: **User**, **Photo**, **Day**

Predicates:

influencer: **User**
sees: **User**×**Photo**
follows: **User**×**User**
date: **Photo**×**Day**

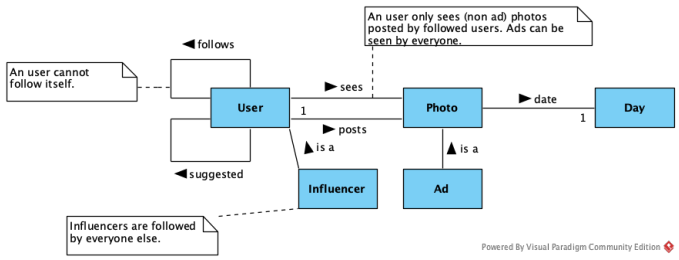
ad: **Photo**
posts: **User**×**Photo**
suggested: **User**×**User**

Example: Instagram



- Codification of specialization relationships. (nothing to add)
- Disjunction of specialization entities. (nothing to add)
- Typing associations that involve subsets. (nothing to add)

Example: Instagram



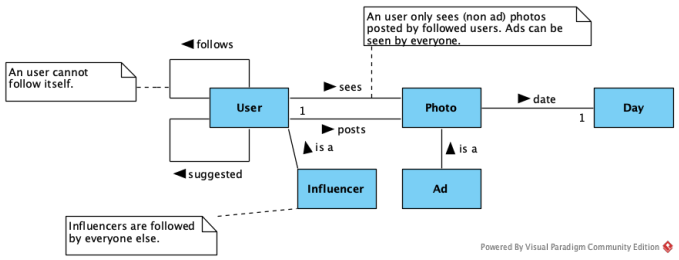
- Multiplicity restrictions on associations. (posts)
  - every photo has to be posted by some user

    $(\forall y : \mathbf{Photo}.\, \exists x : \mathbf{User}.\, \mathsf{posts}(x, y))$
  - every photo is posted by no more than one user

    $(\forall x, y : \mathbf{User}.\, \forall z : \mathbf{Photo}.\, \mathsf{posts}(x, z) \wedge \mathsf{posts}(y, z) \rightarrow x = y)$

  Therefore, every photo is posted by a single user.

Example: Instagram



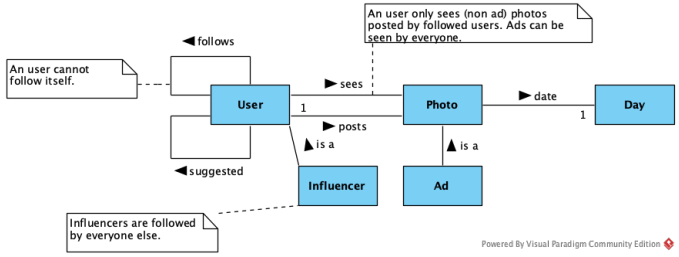
- Multiplicity restrictions on associations. (date)
  - every photo has a date

    $(\forall x : \mathbf{Photo}.\, \exists y : \mathbf{Day}.\, \mathsf{date}(x, y))$
  - every photo has no more than one date

    $(\forall x : \mathbf{Photo}.\, \forall y, z : \mathbf{Day}.\, \mathsf{date}(x, y) \wedge \mathsf{date}(x, z) \rightarrow y = z)$

  Therefore, every photo has a unique date.

Example: Instagram

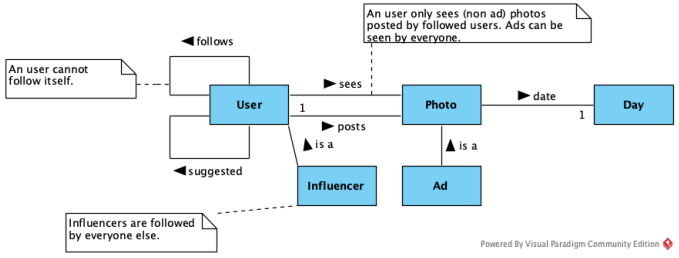


- Restrictions annotated informally.
  - *An user cannot follow itself.*

    $\forall x : \mathbf{User}.\, \neg\mathsf{follows}(x, x)$
  - *Influencers are followed by everyone else.*

    $\forall y : \mathbf{User}.\, \mathsf{influencer}(y) \rightarrow (\forall x : \mathbf{User}.\, x \neq y \rightarrow \mathsf{follows}(x, y))$

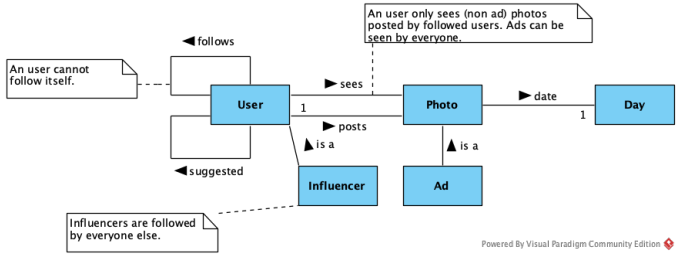
Example: Instagram



- Restrictions annotated informally.
  - *An user only sees (non ad) photos posted by followed users. Ads can be seen by everyone.*

$\forall x : \mathbf{User}.\, \forall y : \mathbf{Photo}.\, \mathsf{sees}(x, y) \rightarrow$
$(\mathsf{ad}(y) \vee (\forall z : \mathbf{User}.\, \mathsf{posts}(z, y) \rightarrow \mathsf{follows}(x, z)))$

Example: Instagram

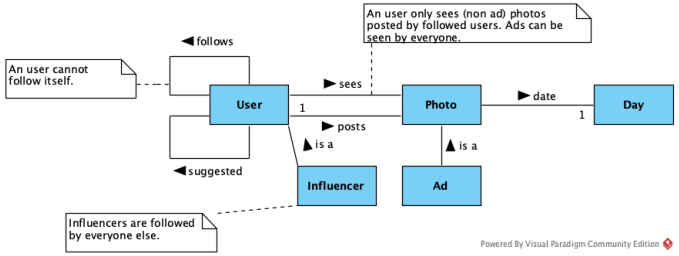


Now write formulas to describe the following properties:

- *Influencers post every day.*

$$\forall x : \mathbf{User}.\ \mathsf{influencer}(x) \rightarrow \forall y : \mathbf{Day}.\ \exists z : \mathbf{Photo}.\ \mathsf{posts}(x, z) \wedge \mathsf{date}(z, y)$$

Example: Instagram

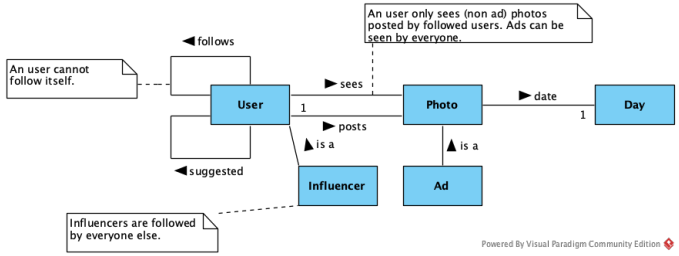


Now write formulas to describe the following properties:

- *If an user posts an ad then all its posts should be labelled as ads.*

$$\forall x : \mathbf{User}.\ (\exists y : \mathbf{Photo}.\ \mathsf{posts}(x, y) \wedge \mathsf{ad}(y)) \rightarrow (\forall z : \mathbf{Photo}.\ \mathsf{posts}(x, z) \rightarrow \mathsf{ad}(z))$$

Example: Instagram



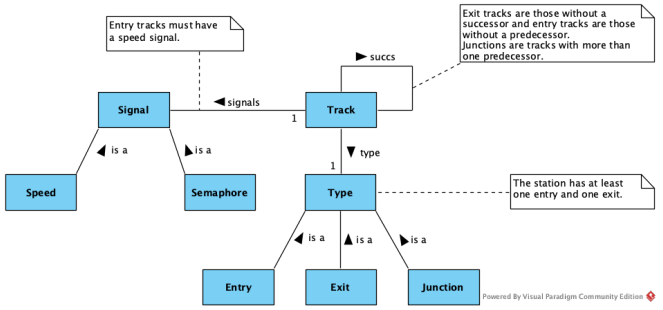
Now write formulas to describe the following properties:

- *Suggested are other users followed by followers but not yet followed.*

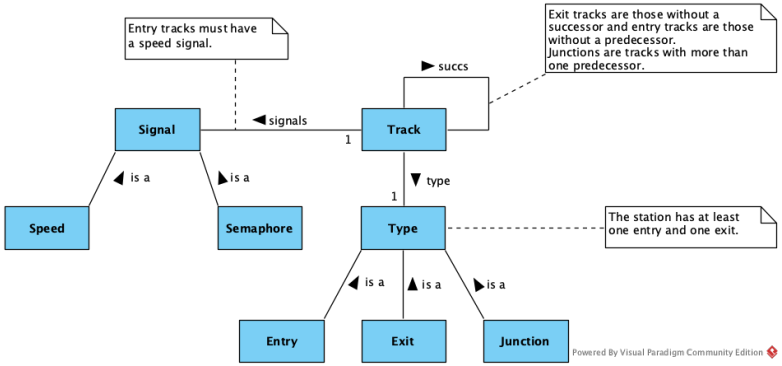
$$\forall x, y : \mathbf{User}.\ \mathsf{suggested}(x, y) \ \leftrightarrow$$
$$(x \neq y \wedge (\exists z : \mathbf{User}.\ \mathsf{follows}(x, z) \wedge \mathsf{follows}(z, y) \wedge \neg \mathsf{follows}(x, y)))$$

Example: Train station

In simplistic terms, the layout of a train station is composed of tracks that are connected to each other. A track can have more than one successor, namely if it ends in a railway switch. A switch can also be used to form a junction between two or more tracks. Trains arrive at the station through entry tracks and leave the station through exit tracks. The interlocking system is responsible for ensuring the safe flow of the trains through a station. A key part of the interlocking are signals. Possible signals are semaphores and speed limits.



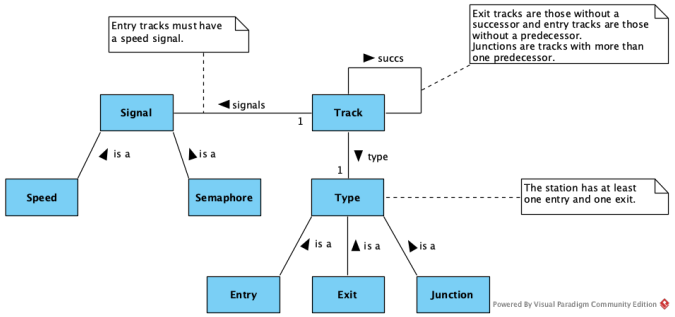
Example: Train station



Types: **Signal**, **Track**, **Type**
Constants: Entry, Exit, Junction: **Type**
Predicates:

speed: **Signal** semaphore: **Signal**
signals: **Track**×**Signal** succs: **Track**×**Track**
type: **Track**×**Type**

Example: Train station

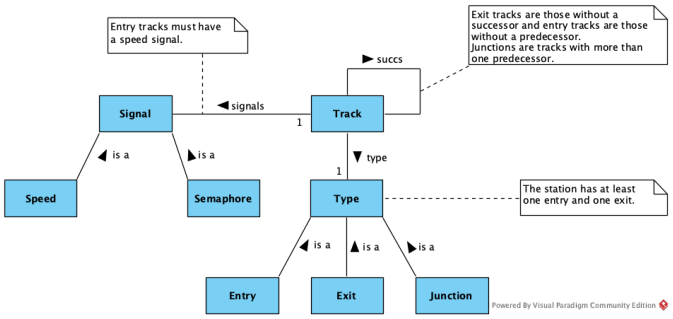


- Disjunction of specialization entities.

$$\text{Entry} \neq \text{Exit} \wedge \text{Entry} \neq \text{Junction} \wedge \text{Exit} \neq \text{Junction}$$

$$\forall x : \mathbf{Signal}.\, \text{speed}(x) \rightarrow \neg\text{semaphore}(x)$$

Example: Train station



- Multiplicity restrictions on associations. (signals)
  - every signal is in a track

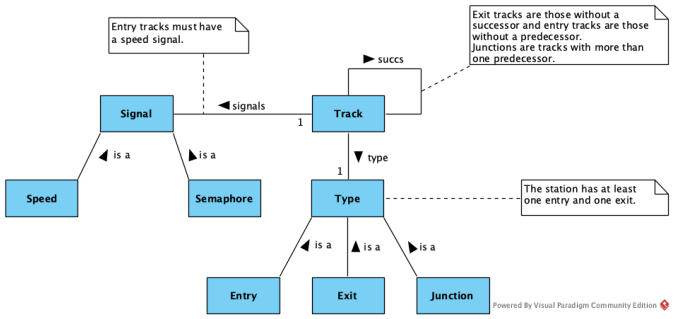
$$(\forall s : \mathbf{Signal}.\, \exists y : \mathbf{Track}.\, \text{signals}(y, s))$$

  - every signal has no more than one track

$$(\forall x, y : \mathbf{Track}.\, \forall z : \mathbf{Signal}.\, \text{signals}(x, z) \wedge \text{signals}(y, z) \rightarrow x = y)$$

Therefore, every signal is associated to one unique track.

Example: Train station



- Multiplicity restrictions on associations. (type)
  - every tract has a type

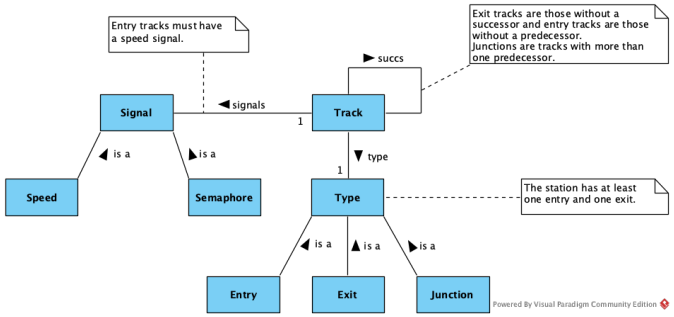
$$(\forall x : \mathbf{Track}.\, \exists y : \mathbf{Type}.\, \text{type}(x, y))$$

  - every track has no more than one type

$$(\forall x : \mathbf{Track}.\, \forall y, z : \mathbf{Type}.\, \text{type}(x, z) \wedge \text{type}(x, y) \rightarrow z = y)$$

Therefore, every track has a unique type.

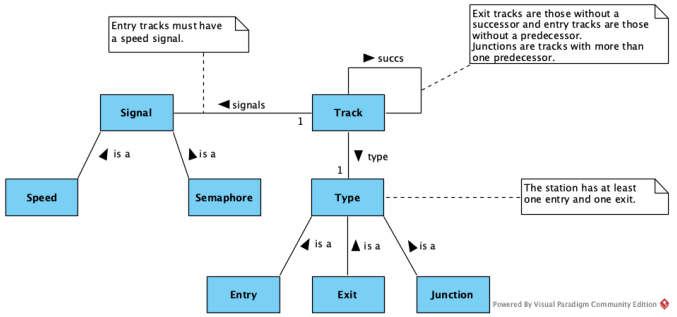
Example: Train station



- Restrictions annotated informally.
  - *Entry tracks must have a speed signal.*

    $\forall x : \mathbf{Track}.\ \mathsf{type}(x, \mathsf{Entry}) \rightarrow \exists y : \mathbf{Signal}.\ \mathsf{signals}(x, y) \wedge \mathsf{speed}(y)$

Example: Train station

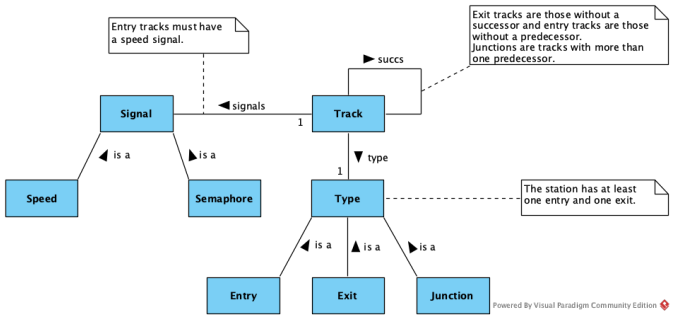


- *The station has at least one entry and one exit.*

  $\exists x : \mathbf{Track}.\ \mathsf{type}(x, \mathsf{Entry})$

  $\exists x : \mathbf{Track}.\ \mathsf{type}(x, \mathsf{Exit})$

Example: Train station



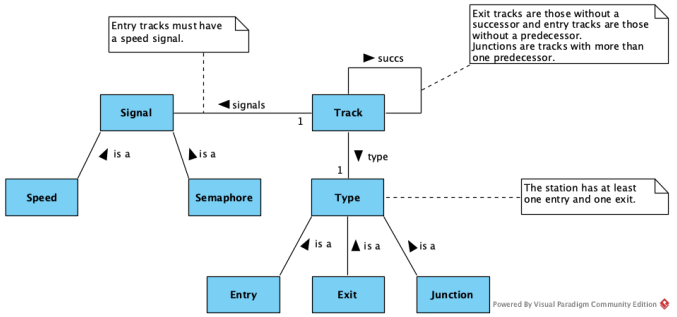
- *Exit tracks are those without a successor and entry tracks are those without a predecessor. Junctions are tracks with more than one predecessor.*

  $\forall x : \mathbf{Track}.\ \mathsf{type}(x, \mathsf{Exit}) \ \leftrightarrow\ \forall y : \mathbf{Track}.\ \neg\mathsf{succs}(x, y)$

  $\forall x : \mathbf{Track}.\ \mathsf{type}(x, \mathsf{Entry}) \ \leftrightarrow\ \neg\exists y : \mathbf{Track}.\ \mathsf{succs}(y, x)$

  $\forall x : \mathbf{Track}.\ \mathsf{type}(x, \mathsf{Junction}) \ \leftrightarrow$
  $\exists y, z : \mathbf{Track}.\ \mathsf{succs}(y, x) \wedge \mathsf{succs}(z, x) \wedge y \neq z$

Example: Train station

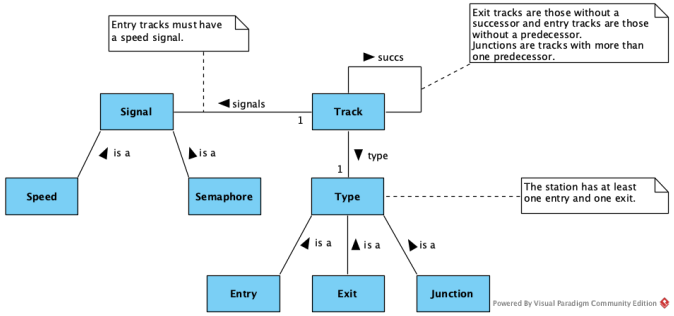


Now write formulas to describe the following properties:

- *Tracks that precede junctions must have semaphores.*

  $\forall x, y : \mathbf{Track}.\ \mathsf{succs}(x, y) \wedge \mathsf{type}(y, \mathsf{Junction}) \rightarrow$
  $\exists s : \mathbf{Signal},\ \mathsf{signals}(x, s) \wedge \mathsf{semaphore}(s)$

Example: Train station

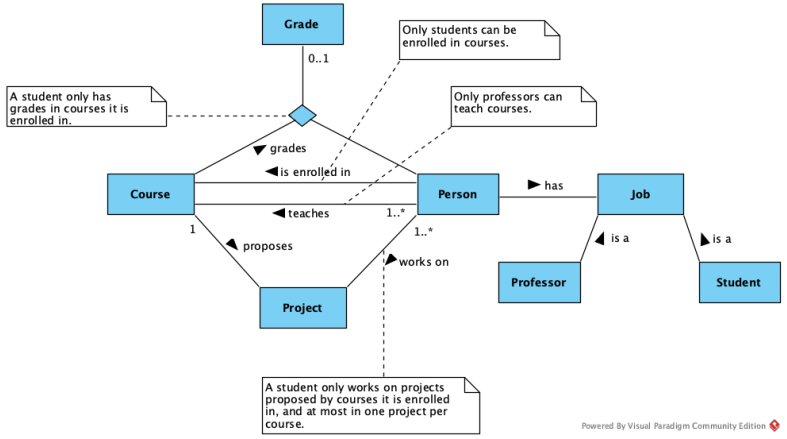


Now write formulas to describe the following properties:

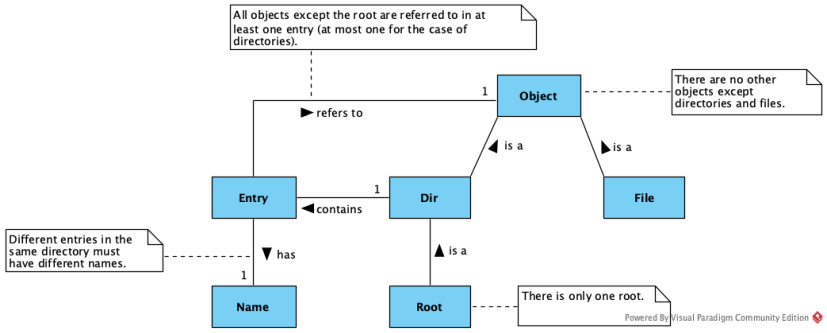
- *If the train station have more than one entry track, then it must have more that two exit tracks.*

$$(\exists x, y : \mathbf{Track}.\ \mathsf{type}(x, \mathsf{Entry}) \wedge \mathsf{type}(y, \mathsf{Entry}) \wedge x \neq y) \rightarrow$$
$$(\exists x, y, z : \mathbf{Track}.\ \mathsf{type}(x, \mathsf{Exit}) \wedge \mathsf{type}(y, \mathsf{Exit}) \wedge \mathsf{type}(z, \mathsf{Exit}) \wedge$$
$$x \neq y \wedge x \neq z \wedge z \neq y)$$

Example: Courses



Example: File system



Example: Production line