

Type Objects

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

ADAP C08

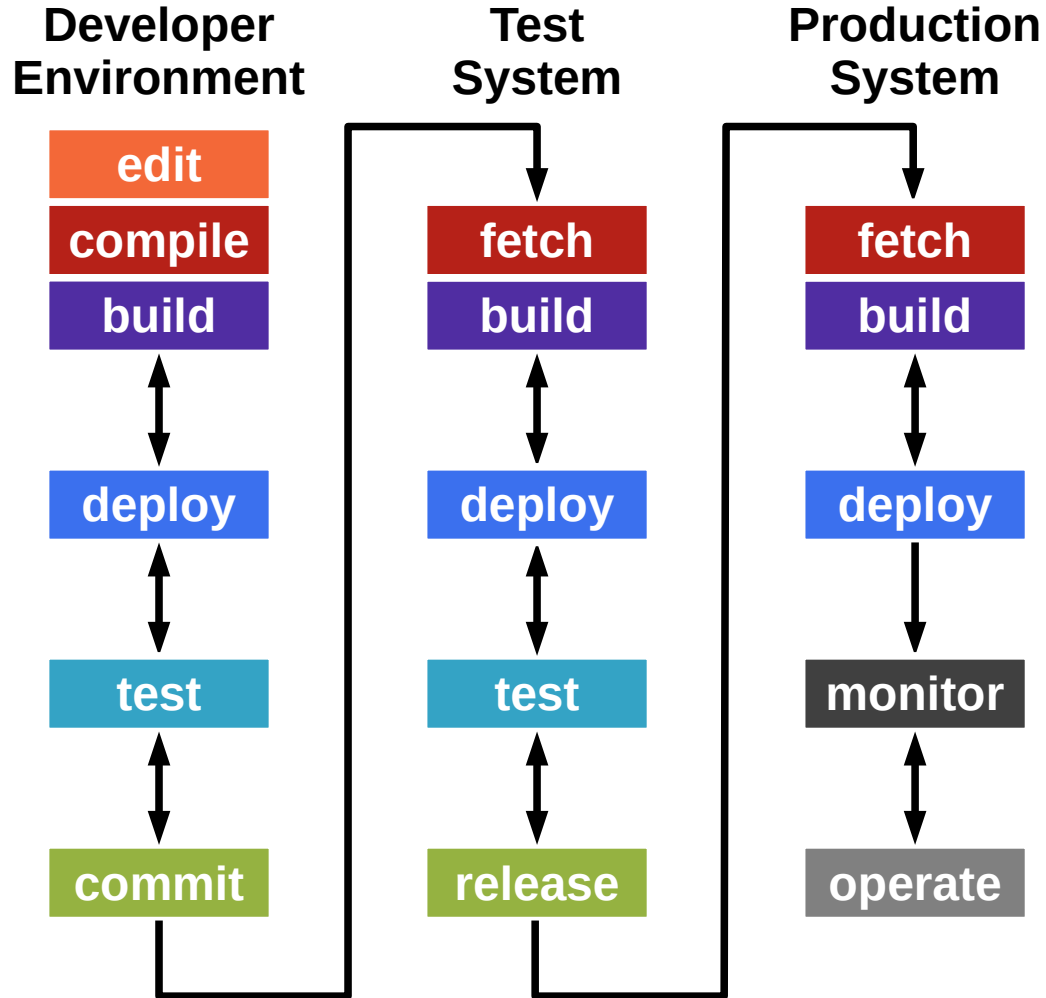
Licensed under [CC BY 4.0 International](https://creativecommons.org/licenses/by/4.0/)

Agenda

1. Design-time vs. run-time
2. Model (type) vs. instance
3. Type object pattern
4. Type object examples
5. Type object hierarchies
6. Meta-object protocols
7. UML-based modeling

1. Design-Time vs. Run-Time

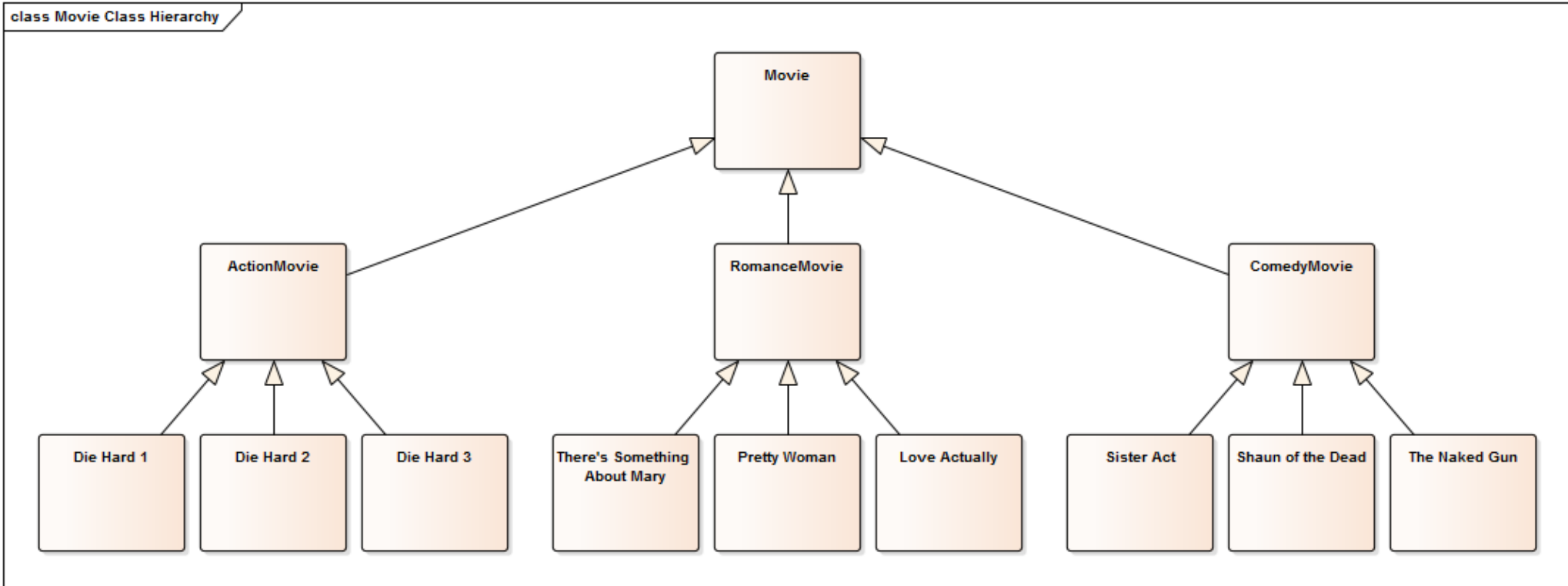
Design-Time vs. Test-Time vs. Run-Time



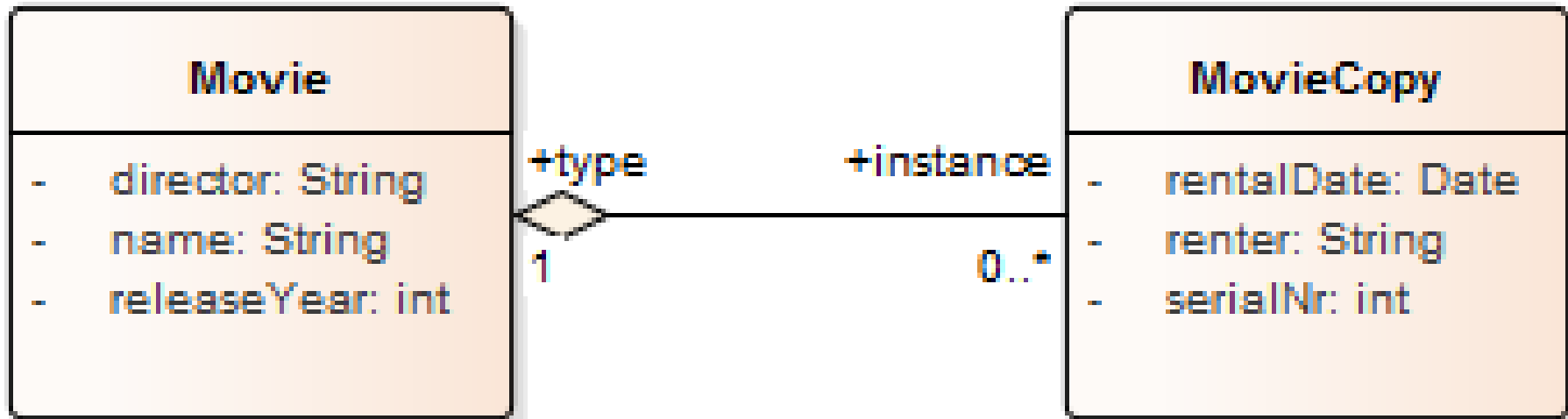
- **Design-time**
 - Change classes
- **Test-Time**
 - Find and file bugs
- **Run-time**
 - Change objects
 - Class loading
 - Configuration
 - Execution

2. Model (Type) vs. Instance

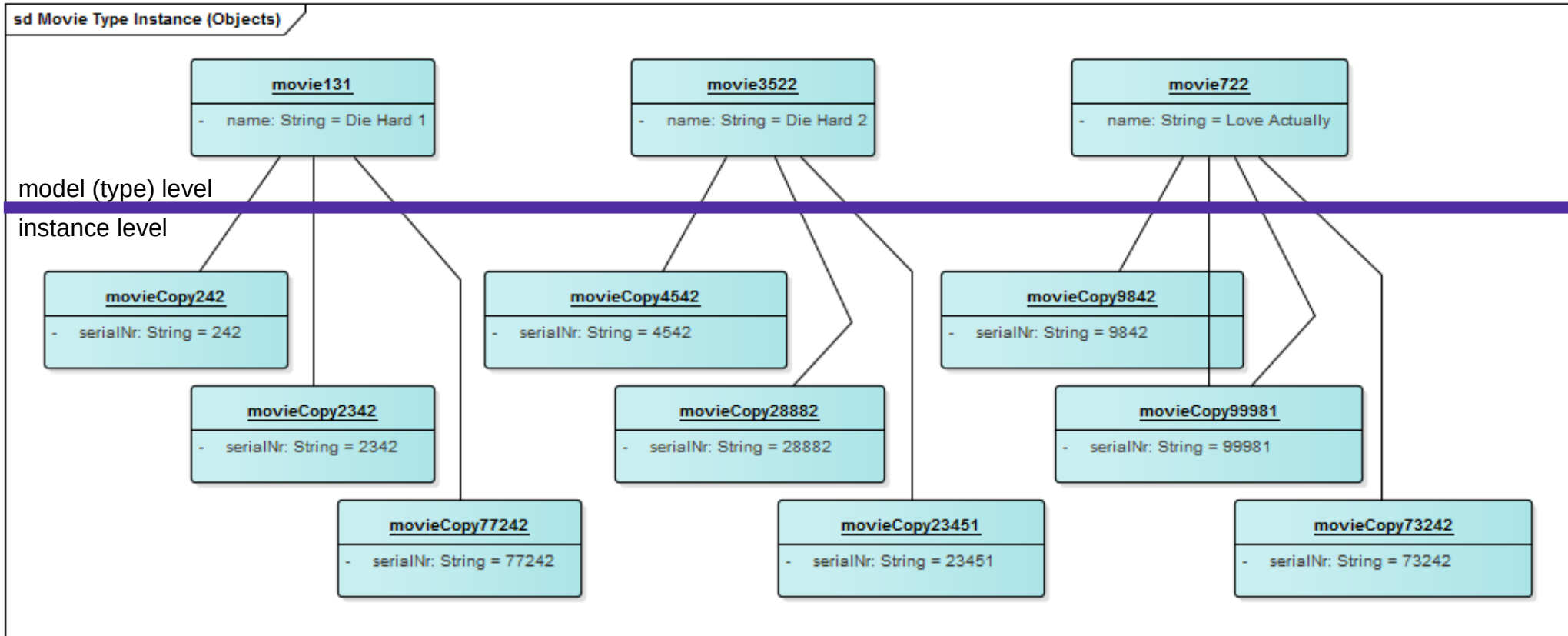
Exploding Class Hierarchies



class Movie Type Instance



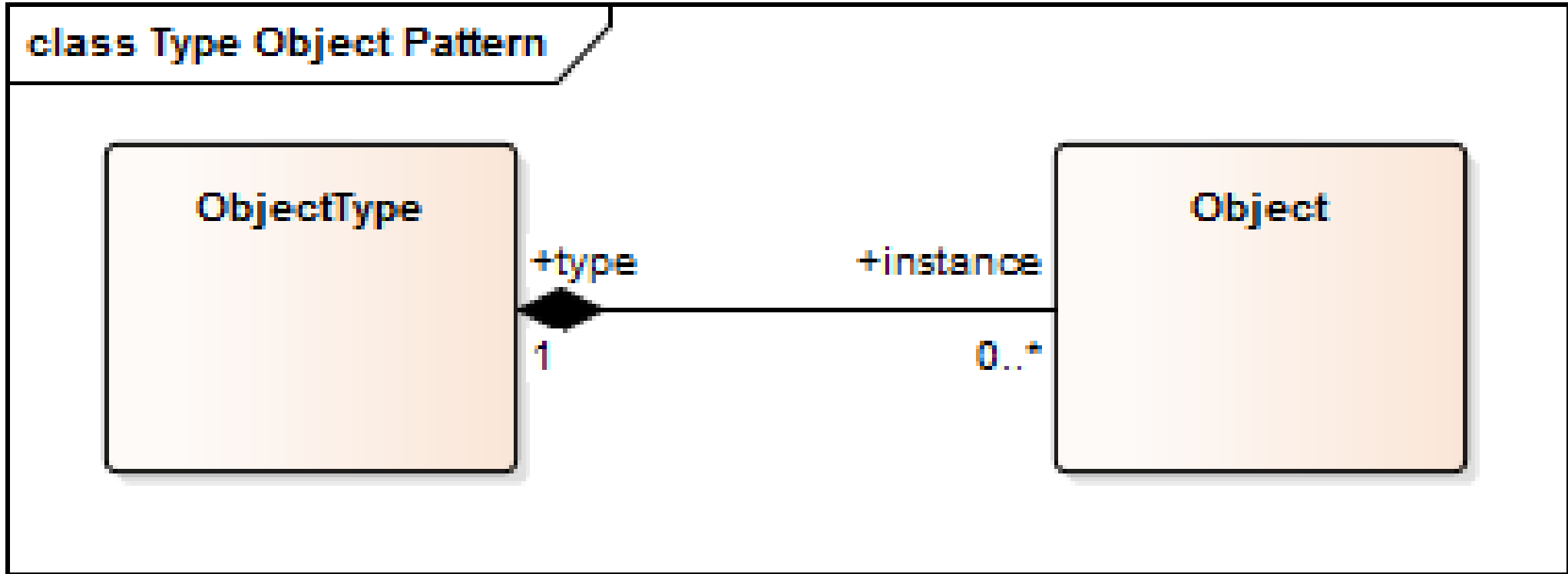
Run-Time / Instances



3. Type Object Pattern

Decouple instances from their classes so that those classes can be implemented as instances of a class. Type Object allows new “classes” to be created dynamically at runtime, lets a system provide its own type-checking rules, and can lead to simpler, smaller systems.

Structure of Type Object Pattern



Collaborations of Type Object Pattern

- Object
 - Provides instance specific functionality
 - Delegates type-specific requests to type object
- ObjectType
 - Handles type-common requests for instances
 - May create and/or manage its instances

Benefits of Type Object Pattern

- Reduces an exploding class hierarchy to two classes
- Allows for managing new classes (types) at runtime

Downsides of Type Object Pattern

- Increased run-time complexity
 - Makes code more difficult to read
 - Makes debugging more difficult

4. Type Object Examples

Examples of Type Object Pattern

- Object and Class
 - `java.lang.Object`: base object class (instances)
 - `java.lang.Class`: Java object type-object-class
- Flower and FlowerType
 - `org.wahlzeit.flowers.Flower`: flower object class (instances)
 - `org.wahlzeit.flowers.FlowerType`: flower type-object-class
- PersonRole and PersonRoleType
 - `com.app.model.PersonRole`: person-role class
 - `com.app.model.PersonRoleType`: person-role type-object-class

Quiz: Defining Keyboard Models [1]

- You are developing software for configuring computers. You are implementing a Keyboard class to represent a keyboard that a customer might choose. However, there are many types of keyboards available and new types keep coming up.

Using the Type Object pattern, how would you design the Keyboard class in Java to make it easy to introduce new keyboard types later on?

Select all good design decisions.

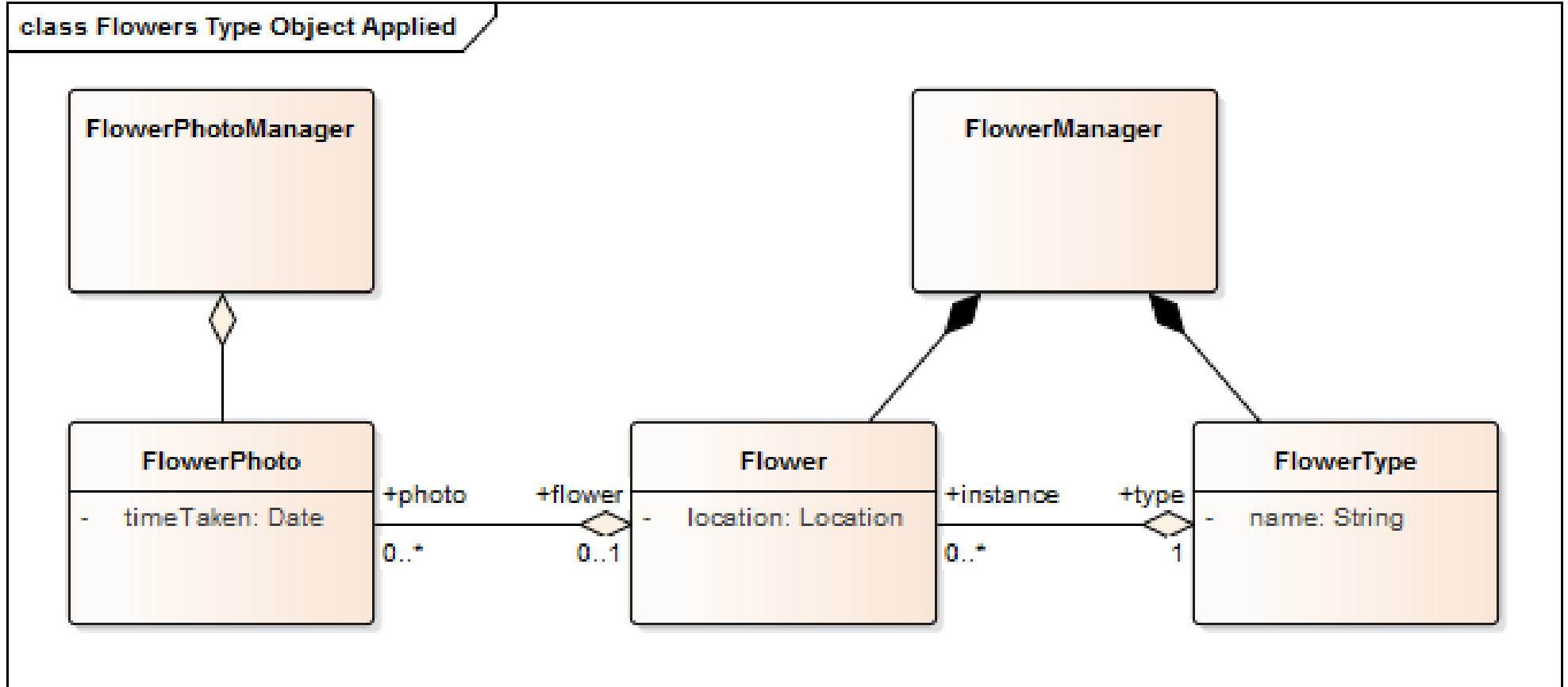
- The Keyboard class defines two string attributes, model and make.
- A separate KeyboardType class defines two string attributes, model and make.
- The Keyboard class defines a reference to a KeyboardType class.
- A KeyboardType class defines a collection of references to Keyboard objects.

[1] Model and make are (common) synonyms for types.

Answer: Defining Keyboard Models

- Using the Type Object pattern, how would you design the Keyboard class in Java to make it easy to introduce new keyboard types later on?
 - The Keyboard class defines two string attributes, model and make.
 - No. This type information belongs into a Type Object class.
 - A separate KeyboardType class defines two attributes, model and make.
 - Yes: You need a KeyboardType class to collect type information.
 - The Keyboard class defines a reference to a KeyboardType class.
 - Yes. A Keyboard object needs to access a KeyboardType object.
A direct reference is the easiest way to access the object.
 - A KeyboardType class defines a collection references to Keyboard objects.
 - No. It is unusual to make the type object track its instances.
If anything, you'll use a Manager object for this task.

Type Object Applied to Flowers



Creating Flower Instances

```
public Flower FlowerManager#createFlower(String typeName) {  
    assertIsValidFlowerTypeName(typeName);  
    FlowerType ft = getFlowerType(typeName);  
    Flower result = ft.createInstance(...);  
    flowers.put(result.getId(), result);  
    return result;  
}
```

```
public Flower FlowerType#createInstance() {  
    return new Flower(this);  
}
```

```
protected FlowerType Flower#flowerType = null;  
  
public Flower#Flower(FlowerType ft) {  
    flowerType = ft;  
}
```

5. Type Object Hierarchies

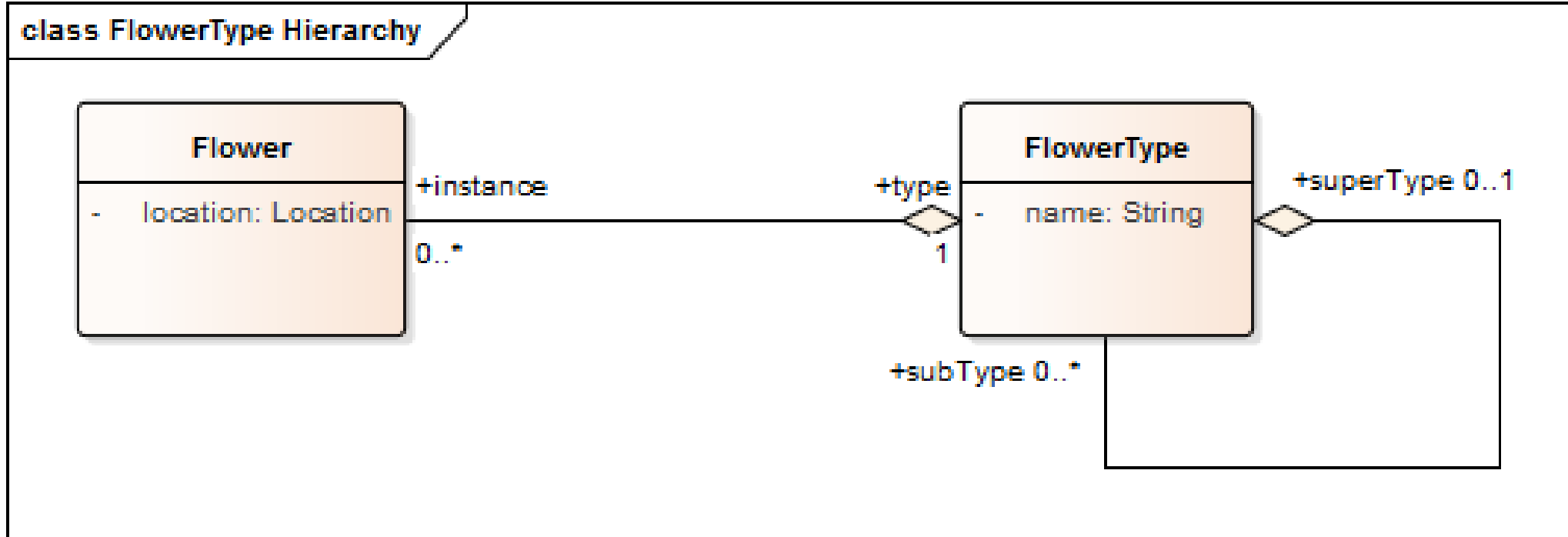
Quiz: Type Object Hierarchy



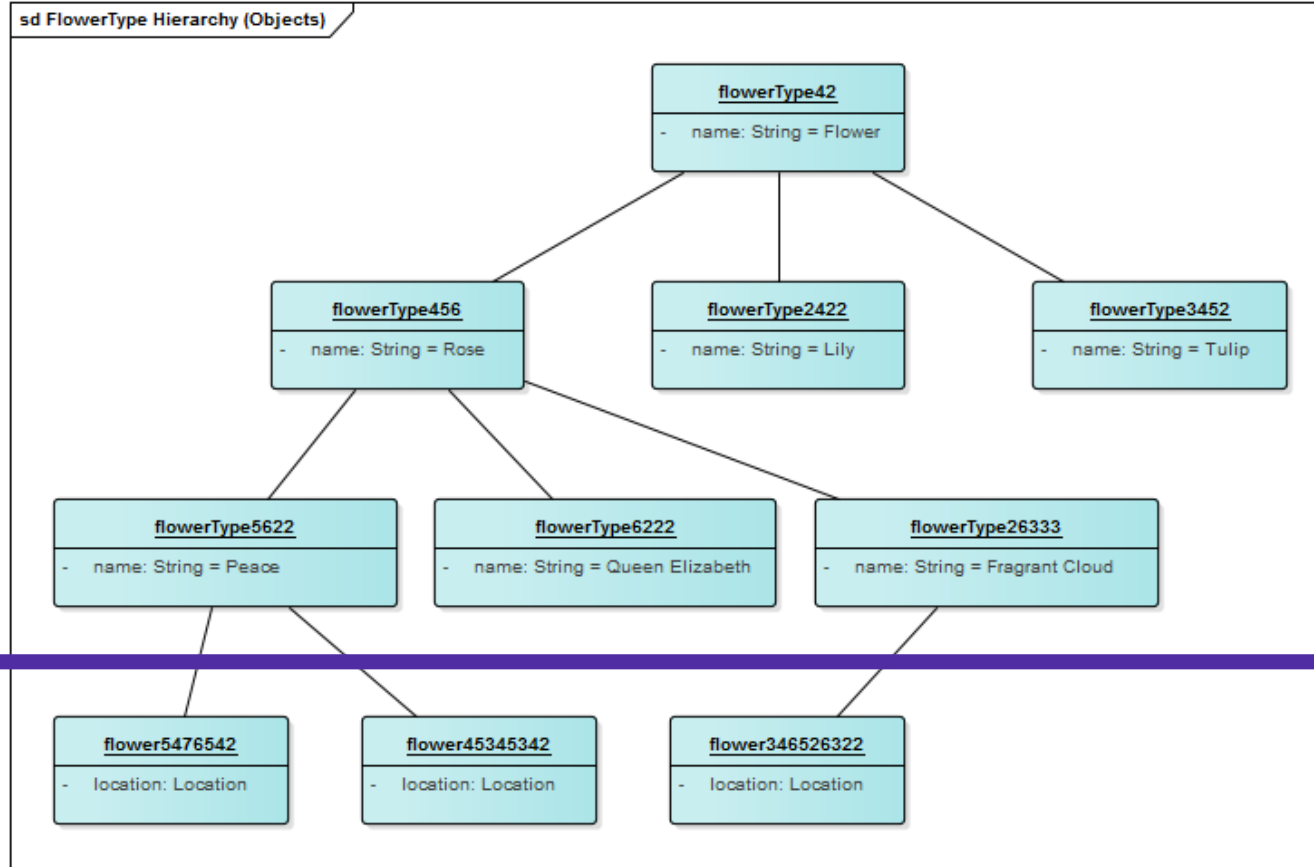
Plants are classified (in a hierarchy) according to the *International Code of Nomenclature for Cultivated Plants*.
How to represent a type hierarchy for flowers in Flow-ers?

Flower families photo courtesy of Wikipedia

Answer 1 / 2: Type Object Hierarchy



Answer 2 / 2: Type Object Hierarchy



model (type) level

instance level

Implementing the FlowerType Hierarchy

```
public class FlowerType extends DataObject {
    protected FlowerType superType = null;
    protected Set<FlowerType> subTypes = new HashSet<FlowerType>();

    public FlowerType getSuperType() {
        return superType;
    }

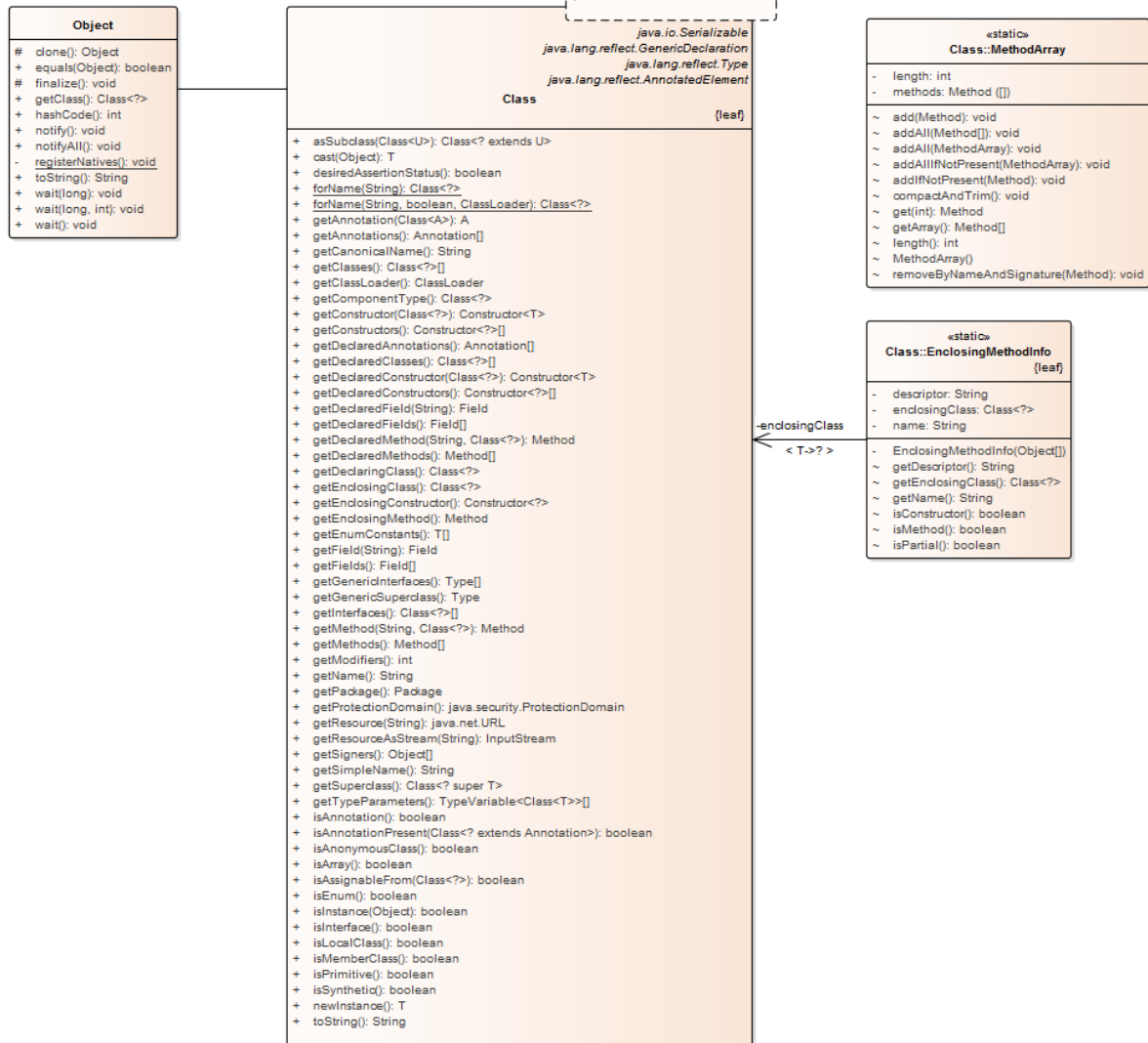
    public Iterator<FlowerType> getSubTypeIterator() {
        return subTypes.iterator();
    }

    public void addSubType(FlowerType ft) {
        assert (ft != null) : "tried to set null sub-type";
        ft.setSuperType(this);
        subTypes.add(ft);
    }

    ...
}
```

Using a FlowerType Object

```
public class FlowerType {  
    public boolean hasInstance(Flower flower) {  
        assert (flower != null) : "asked about null object";  
  
        if (flower.getType() == this) {  
            return true;  
        }  
  
        for (FlowerType type : subTypes) {  
            if (type.hasInstance(flower)) {  
                return true;  
            }  
        }  
  
        return false;  
    }  
    ...  
}
```

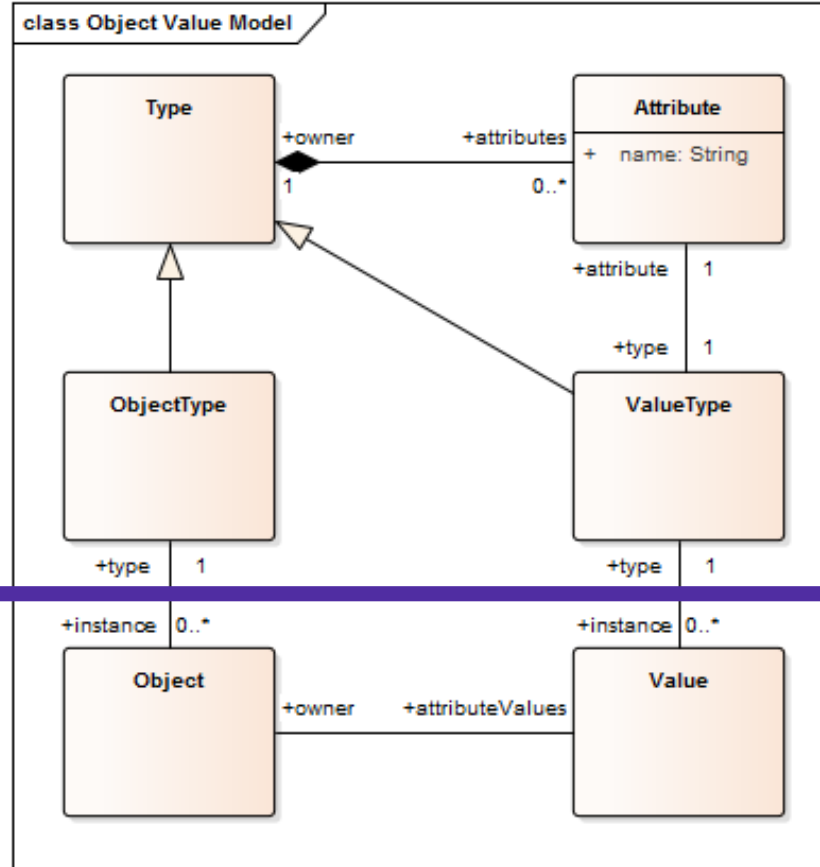


6. Meta-Object Protocols

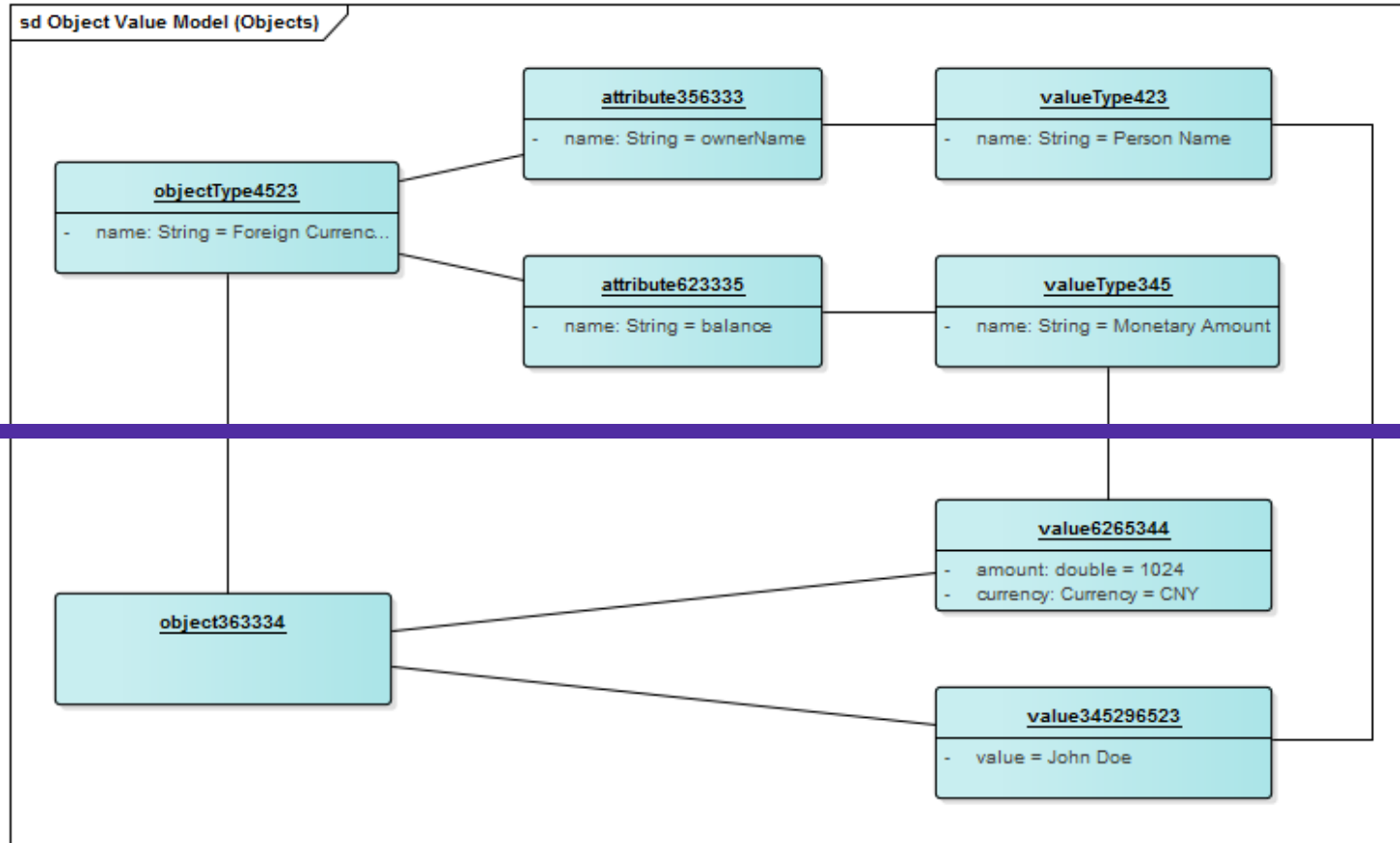
Simple System Model

model (type) level

instance level



Example of Simple System Model



model (type) level
instance level

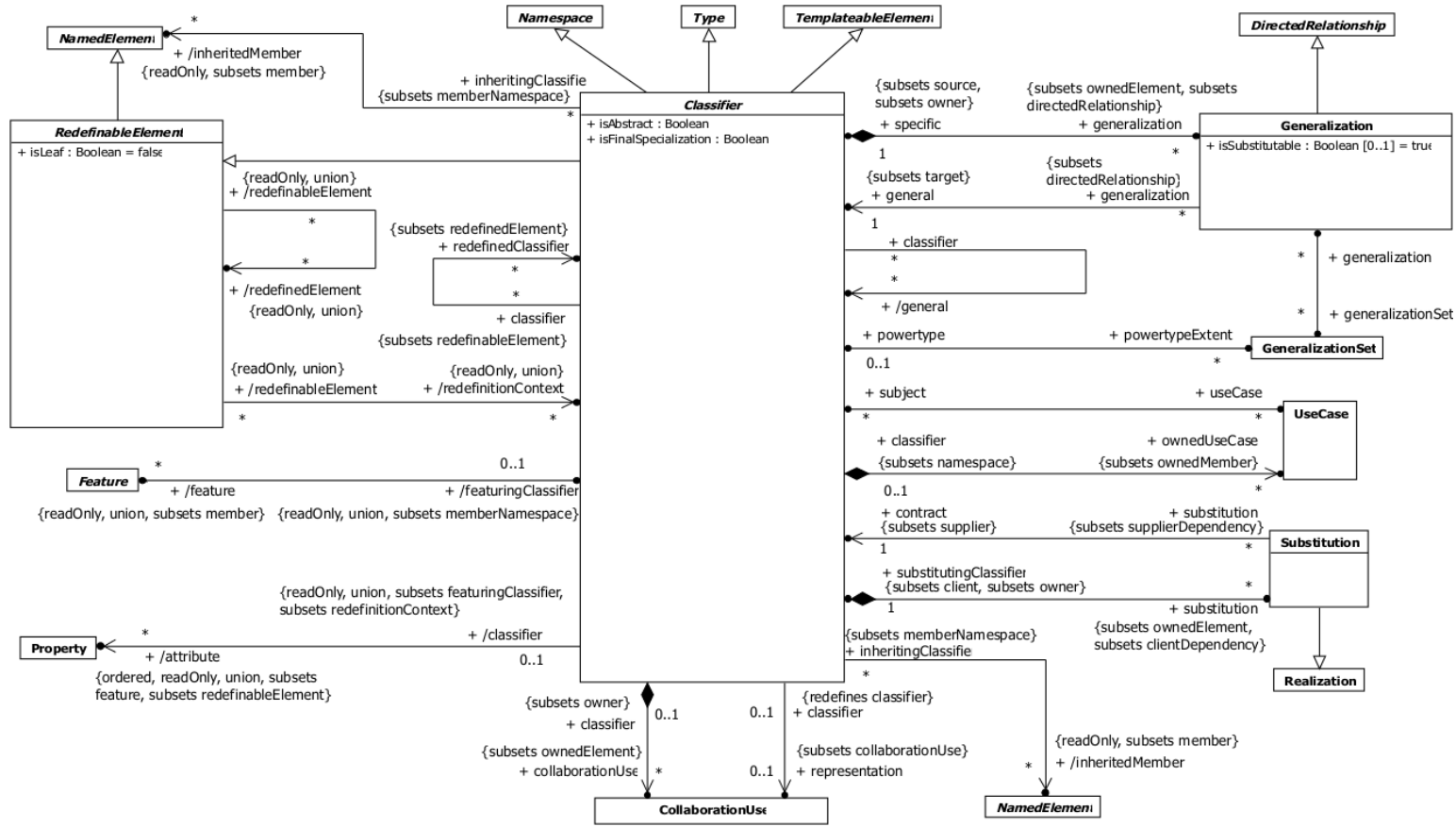
- Metaclass, class and object
 - Usually used in absolute terms, covering levels M2, M1, M0
 - Metaclass = element of M2 level (language level)
 - Class = element of M1 level (model level)
 - Object = element of M0 level (object/instance/run-time level)
- Meta-object and base-object
 - Usually use as relative terms: the meta-object describes the base-object
 - A meta-object can be the base-object for another meta-object
- Type and instance
 - Usually used as relative terms, similar to meta and base-object

Meta-Object Protocols (MOPs)

- Introspection
 - Provide information about base objects
- Intercession
 - Manipulate structure and behavior of base objects

7. UML-based Modeling

The Classifier Part of the UML Metamodel



Java, UML, Flowers

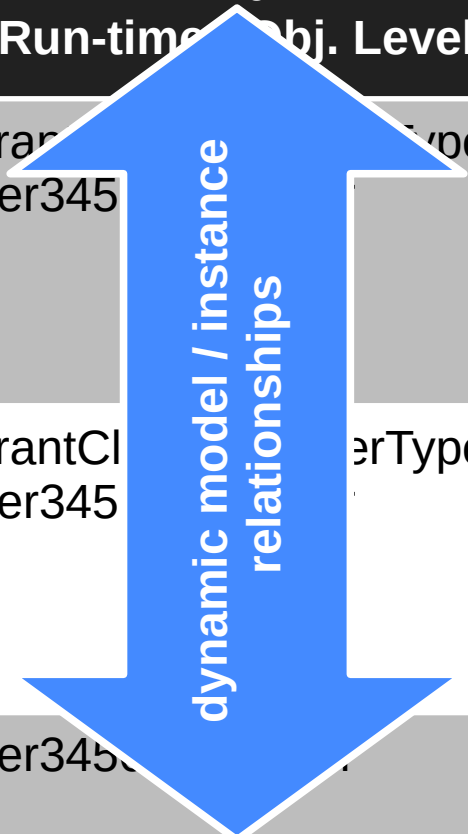
| M2 Language Level | M1 Model / Code Level | M0 Run-time / Obj. Level |
|--|--|---|
| java.lang.Class ... | java.lang.Object flowers.FlowerPhoto flowers.FlowerType flowers.Flower ... | fragrantCloud : FlowerType flower34563 : Flower ... |
| uml.Classifier uml.Generalization uml.Stereotype uml.Component ... | flowers.FlowerPhoto flowers.FlowerType flowers.Flower ... | fragrantCloud : FlowerType flower34563 : Flower ... |
| flowers.FlowerType ... | flowers.Flower : FlowerType fragrantCloud : FlowerType ... | flower34563 : Flower ... |

Model / Instance Relationships (Static)

| M2 Language Level | M1 Model / Code Level | M0 Run-time / Obj. Level |
|--|---|---|
| java.lang.Class ... | java.lang.Object flowers.FlowerPhoto flowers.FlowerType flowers.Flower | fragrantCloud : FlowerType flower34563 : Flower ... |
| uml. uml.Generalization uml.Stereotype uml.Component ... | flowers.Flower ... | ... |
| flowers.FlowerType ... | flowers.Flower : FlowerType fragrantCloud : FlowerType ... | flower34563 : Flower ... |

Model / Instance Relationships (Dynamic)

| M2 Language Level | M1 Model / Code Level | M0 Run-time Obj. Level |
|--|--|---|
| java.lang.Class ... | java.lang.Object flowers.FlowerPhoto flowers.FlowerType flowers.Flower ... | fragrantCloud : FlowerType flower345 : FlowerType ... |
| uml.Classifier uml.Generalization uml.Stereotype uml.Component ... | flowers.FlowerPhoto flowers.FlowerType flowers.Flower ... | fragrantCloud : FlowerType flower345 : FlowerType ... |
| flowers.FlowerType ... | flowers.Flower : FlowerType fragrantCloud : FlowerType ... | flower345 : FlowerType ... |



Summary

1. Design-time vs. run-time
2. Model (type) vs. instance
3. Type object pattern
4. Type object examples
5. Type object hierarchies
6. Meta-object protocols
7. UML-based modeling

Thank you! Questions?

dirk.riehle@fau.de – <https://oss.cs.fau.de>

dirk@riehle.org – <https://dirkriehle.com> – [@dirkriehle](#)

Legal Notices

- License
 - Licensed under the **CC BY 4.0 International** License
- Copyright
 - © 2012-2022 Dirk Riehle, some rights reserved