Programming in Java

5. Inheritance

Introduction

- The focus of this module is the concepts of inheritance, abstraction and how these are implemented in Java
- There a more background material than we can cover in class
  - There is additional content in the extras folder in the lab
- The main Java related topics to be covered are:
  - The difference between implementing specialization versus generalization inheritance
  - Inheritance hierarchies of Java classes
  - Abstract classes
  - Interfaces as types

Domain versus Design Classes

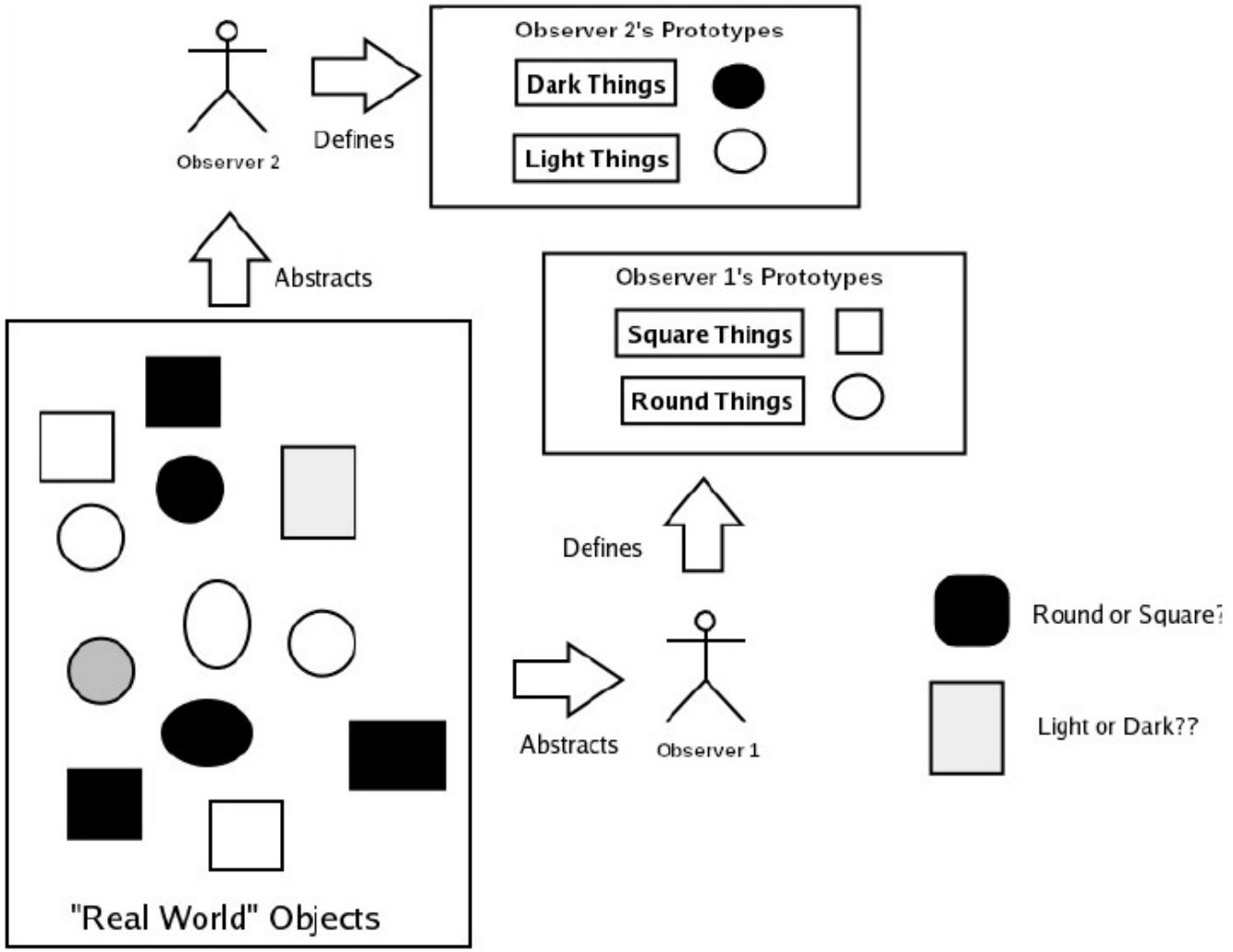
- Domain Classes
  - These correspond to the types that have conceptual reality in the real-world application domain.
  - Domain classes use prototypes that we have to discover by investigation and analysis
  - Domain definitions tend to be fluid and fuzzy
- Design Classes
  - These are the classes we define to build our system
  - These are the classes that exist in the software and do not necessarily copy the domain classes in our problem domain
  - However, the choice of design classes (Java packages and classes) are guided by an understanding of the domain
  - This is the practice of *Domain Driven Design*

The Chen Hierarchy

- Mental Models
  - Information concerning entities and relationships which exist in our minds
  - This tends to be idiosyncratic and varies from person to person
  - Translating these into code is usually a really bad idea
  - Types are represented as prototypes
- Information structure
  - Organization of information in which entities and relationships are represented by data
  - Agreed upon standard models of the world by user communities
    - *“When we say ‘customer’ we mean….”*
  - Defined by clear and unambiguous predicates and value sets
- Data Structures
  - The implementation of the information structure in something like a relational table or Java class
- Chen’s paper is in the extras folder

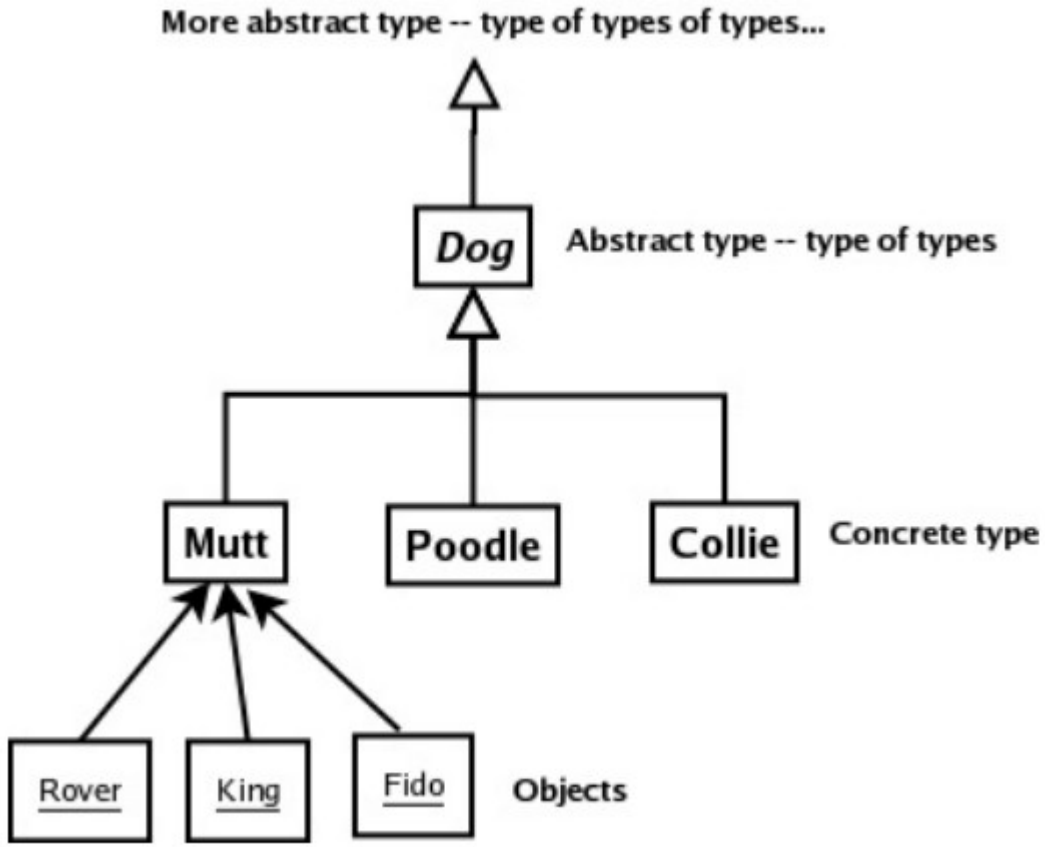
Abstraction

- We perform abstraction, also called generalization, based on three kinds of similarity among objects
  - **Appearance**: Objects that are perceptually similar. e.g. trees, rocks, circles, bangs, stinks
  - **Interaction**: Objects that we interact with in similar ways. e.g. tools, vehicles, food
  - **Relationship**: Objects that exist in similar relationships with other objects. e.g. employees, parents, furniture



Inheritance and Abstract Types

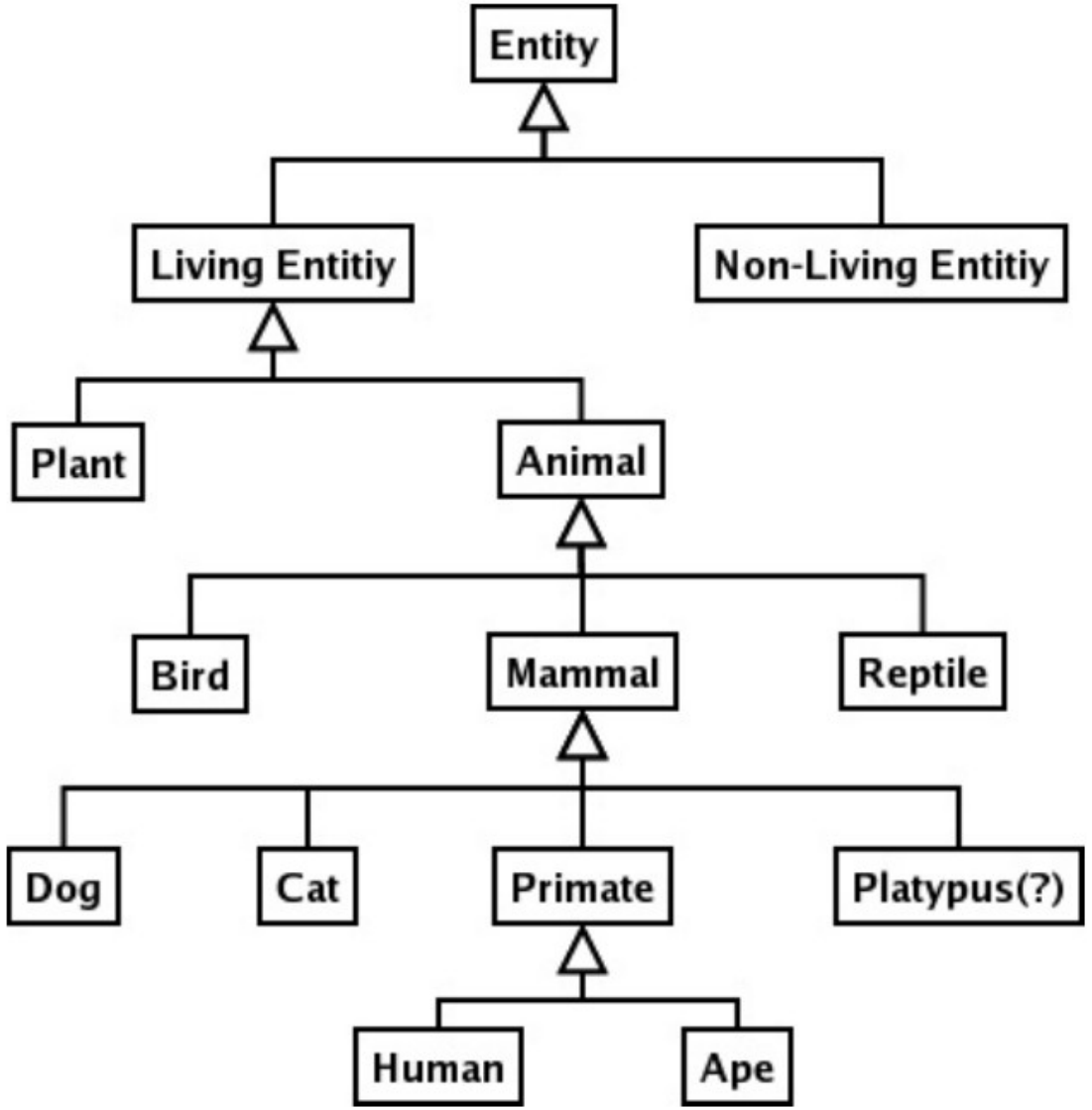
- If a class is an object then we can group classes into classes-of-classes which we call abstract classes
  - This is the start of our abstraction hierarchy
- How do we know something is an abstract class?
  - Because the class description is generally not complete enough for us to specify a well-formed object of that type.
  - Or in plain English objects of only that class just don't exist



Abstract Types

- Abstract classes, and prototypes in general, are often used when recording business rules or system knowledge in a form like a script or user story
  - People tend to store and communicate information as scenarios
  - Details not germane to the point of the story are abstracted away
    - *To get money out of an account, a customer goes to an ATM, swipes their card, enters their PIN, selects withdraw, selects the account and amount, then takes their cash and receipt.*
  - We don’t need to know concrete details like how much they withdrew, where the ATM was, who the customer is – the focus in on the process
- Constructing these abstraction hierarchies allows for efficient information processing and inferencing

Abstract Types



Generalization and Specialization

- Generalization refers to the process just described
  - We group a set of classes into a super-type or abstract class
  - We can continue to do this to create a hierarchy where only the lowest level are concrete classes
- Specialization is where we take an existing concrete class
  - Create a new sub-type with additional functionality or specialized use
  - There are no abstract types involved
  - For example:
    - *Service dog is a specialization of the concrete class dog*
    - *Dress shoes is a specialization of shoes*
    - *Drive through ATM is a specialization of ATM*

Labs

Inheritance

Interfaces

- An interface defines the set of messages that we can send to a class
  - Corresponds to a real word idea of an interface
    - *Identifies what objects of a specific type can be asked to do by other objects*
  - We can identify what type an object is by the interface it implements
  - All classes can have a public interface made up of the public methods and a package interface made up of package methods
- Java interfaces allow us to define what the public methods for a class should be
  - Often called business interfaces
  - Best practice is to define the interfaces before writing the classes
  - This ensures we implement all the necessary behavior
- Interfaces are preferred to abstract classes
  - They are more efficient
  - They are the preferred way to design code in most programming languages

Interfaces

- With inhertiance, we can only have one super class
- However we can implement as many interfaces as we want
- We can create so-called “marker” interfaces
  - These are empty – they define no methods signature
  - We can use these to create arbitrary groups of classes
  - Useful for collections of heterogeneous types
- We can also extend interfaces the same way we use extends for classes


Labs
Interfaces


Java™