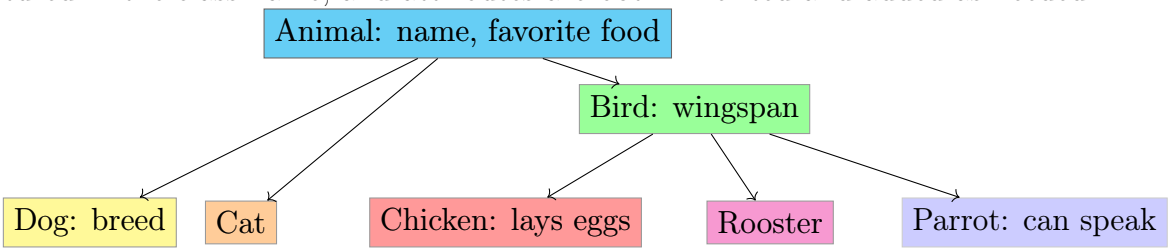
1 Design – short overview

1.1 Overall

Following the dependency inversion pattern the dependencies are mostly declared using interfaces. This allows to test components in a more controlled and isolated manner. For example while a wrapper to random provider just forwards the call to ThreadLocalRandom.current(), it allows to feed particular values in tests. When the app is starting, then the implementations of those types are newed up, and fed to the components.

1.2 Animals via polymorphism

All species of animals are introduced as subclasses of Animal. The name of the species is captured in the class name, and attributes are both inherited and added as needed:



The list of the animals is provided to the rest of the app via an implementation of the interface AnimalsProvider, and thus the constructors of different animals can and are restricted to be package private. An exception is Animal itself, because it is useful to have the possibility to create instances of Animal outside the animals package to test other components in other packages. Since there is no further constructing or copying of the animals in the app, there is no need to override the method equals and hashCode. Memory location based default behaviours identify the animal correctly.

1.3 Day

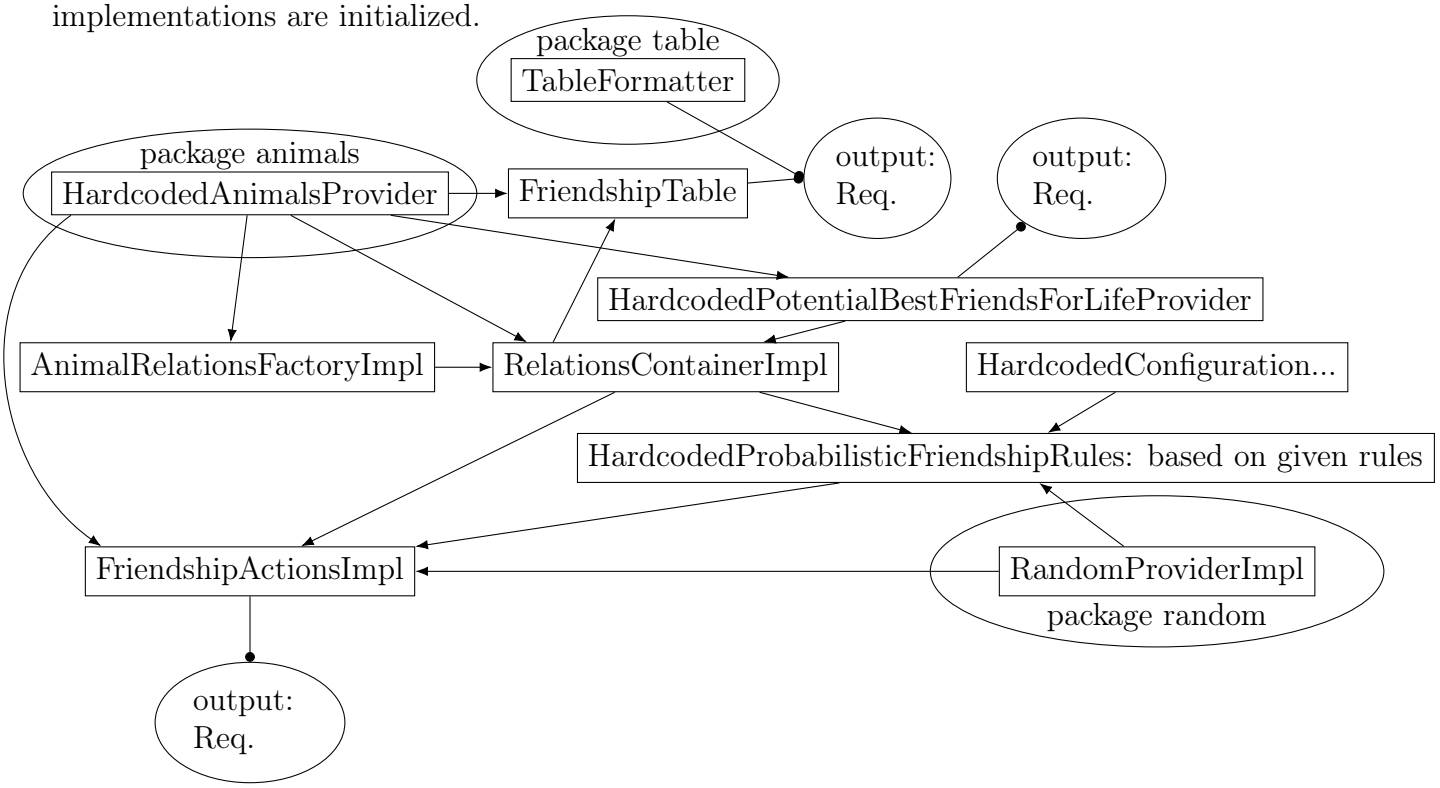
There is a class for day, which interacts with FriendshipActions, Lunch, FriendshipTable, and TableFormatter class. First is used to start a round of loosing friending before noon, and a round of gaining friends after noon. Second is used to provide the eaters grouped by favourite food. In the end of the day the friends table is printed out based on the input from FriendshipTable class, and formatted by TableFormatter class.

1.4 Relations

All friendships are symmetric. If A is a friend of B, then B is a friend of A. This also applies to "(potential) best friend forever" relations.

The symmetry is not rigorously enforced by the underlying implementation in hope that having a separate instance of AnimalRelations for each animal is easier to read. The implementation of RelationsContainer gathers the relations for all animals, and there the symmetry is noted by having UnorderedPair as input parameter for addFriendship and removeFriendship. In the current implementation RelationsContainerImpl there is a method symmetrizeBiConsumerUse which guarantees that both animals in the pair are treated equally.

This package has the following dependencies between the implementations (in a class the dependency is based on interface, but the following graph shows how the current implementations are related). See also the class RelationsStructure in package relations where the implementations are initialized.



Let us consider an animal who has lost a friendship because some other animal has decided to give up their friendship. Can the animal lose any more friends in this round? I went for the implementation that this animal should be considered "safe" from further unfriend actions for this day. I introduced an IteratorWithFeedback (in package generics), and it's two factory method providers: ActionCountsAgainstDailyQuotaOfBothMembersInPair and ActionCountsAgainstDailyQuotaOfOnlyInitiator. For the first version feedback removes both the initiator as well as the responder from the list of active participants. Hence the responder can't initiate unfriend because it is out of the iterator, and also neither can be unfriended by any other animals, because the "safe" animals are removed from the corresponding friend lists.