Using Signatures and Sets: Airport Example

Signatures (reminder)

```
sig A {}
```
set of items A

```
sig A {}
sig B {}
```
disjoint sets A and B (no common members)

```
sig A, B {}
```
same as above

```
sig B extends A {}
```
set B is a subset of A

```
sig B extends A {}
sig C extends A {}
```
B and C are disjoint subsets of A

```
sig B, C extends A {}
```
same as above

```
abstract sig A {}
sig B extends A {}
sig C extends A {}
```
A partitioned by disjoint subsets B and C
(B & C is empty, and A = (B + C))

```
sig B in A {}
```
B is a subset of A – not necessarily disjoint from any other set

```
sig C in A + B {}
```
C is a subset of the union of A and B

```
one sig A {}
lone sig B {}
some sig C {}
```
A is a singleton set
B is a singleton or empty
C is a non-empty set

Signatures with Fields

A signature may include *fields*.

There is an analogy with object oriented programming:

signature ≈ class, field ≈ instance variable (attribute)

```
sig  Aircraft
{
    capacity : Int,              //  a field
    onboard: set Person    // another field
}
```

The O-O analogy is a simplification. We will understand fields more accurately in the next lecture when we learn about relations.

If a is an Aircraft, a.capacity refers to the capacity field of a.

Signatures with fields

```
sig A {f: E}
```
Signature A has a field of type E

```
sig A {
   f1: one  E1,
   f2: lone E2,
   f3: some E3,
   f4: set  E4
}
```
f1 has one unique value
f2 has zero or one value
f3 has one or more values
f4 has any number of values

```
sig A {f, g: E}
```
two fields of the same type

Signatures with Constraints

A signature may have additional *constraints*. These are boolean expression placed within braces after the fields. They may refer directly to the fields. Example:

```
sig  Aircraft
{
    capacity : Int,
    onboard: set Person
}
{
     # onboard <= capacity       // a constraint
}
```

Constraints impose restrictions on the possible models of the specification. The Analyser will only show models that meet the constraints.

Aircraft Boarding Example (revisited)

```
sig  Person { }    // a set of “persons”

sig  Aircraft        // a set of “Aircraft”
{
    capacity : Int,              //  capacity of the aircraft
    onboard: set Person   // the set of people on the aircraft
}
{
     # onboard <= capacity
}
```

The signature Aircraft can be used to model different physical aircraft, or different states of the same physical aircraft.

Predicates

A ***predicate*** is a parameterized boolean expression (like a Java method with boolean return type).

Example:

```
pred full [a:Aircraft] { # a.onboard = a.capacity }
```

We can *run* a predicate in the Alloy Analyser. The Analyser will try to find an example that makes the predicate true.

```
run full
```

Exercise: write a predicate to describe a non-empty Aircraft.

Facts

A ***fact*** is a boolean expression which imposes some additional constraint on our specification.

Example:

```
fact manyPeopleExist { # Person > 2 }
```

If this fact is added to the specification, whenever we run a predicate Alloy will show us only model instances containing at least 2 people.

Assertions

An **assertion** is a boolean expression expressing some property that we think should follow from our specification.

Consider the capacity of an Aircraft. Our specification says it is an Int, but does not say that it has to be greater than 0.

Could an aircraft have a negative capacity? Or does our specification imply that the capacity must be non-negative?

We can write an assertion to check this:

```
assert capacityNonNegative { all a:Aircraft | a.capacity >= 0 }
```

The Analyser can check if there are any counter-examples to this assertion:

```
check capacityNonNegative
```

Predicates, Facts, and Assertions

In a later lecture we will learn more about how to write predicates, facts, and assertions using the full logical language of Alloy.

In another lecture, we will look at the theory of model checking and find out what is really going on when we “run” a predicate or “check” an assertion.

To finish off this lecture, let us look at one important way that predicates are used: to model changes of state.

Modelling changes of state

We can use a predicate to describe an operation that changes the state of a system, such as when a new passenger boards the Aircraft.

Running this predicate will show examples of the operation in action.

```
pred Board
[
    a, a': Aircraft,    // a is the state before boarding, a' the state after
    p:Person            // the person boarding the Aircraft
]
{
    p not in a.onboard                  // precondition
    a'.onboard = a.onboard + p          // postcondition
    a'.capacity = a.capacity            // frame condition
}
```

For you to do: disembarking

```
pred Disembark
[
    // What variables are needed?
]
{
    // What conditions need to be met (preconditions)?
    // How does the state of the system change (postconditions)?
    // What parts of the system are unaffected (frame conditions)?
}
```