# How-to: Inheritance and aggregation

In order to demonstrate the principles of object inheritance and aggregation, let's implement the logic of creating batches based on receipts and production documents. Let's make it so that each new document with a Posted property will automatically generate exactly one new product batch.

Let's update our logic with the notion of a product whose batches will be accounted for:

```
CLASS
Item 'Prod
uct';
name 'Name'
 = DATA IS
TRING[50]
(Item) IN
id;
FORM
items 'Pro
ducts'
    OBJECTS
 i = Item
    PROPER
TIES(i)
name, NEW,
DELETE
;
NAVIGATOR
{
    NEW
items;
}
```

Let's create a **Receipt** class with objects that will indicate the receipt of products:

```
CLASS Receipt 'Arrival';
date 'Date' = DATA DATE
(Receipt) IN id;
item 'Product' = DATA Item
(Receipt);
nameItem 'Product' (Receipt
r) = name(item(r)) IN id;
posted 'Completed' = DATA BOO
LEAN (Receipt);

FORM receipts 'Arrivals'
    OBJECTS r = Receipt
    PROPERTIES(r) date,
nameItem, posted, NEW, DELETE
;
NAVIGATOR {
    NEW receipts;
}
```

For the purposes of this example, let's use a simplified scheme with a single class. In reality, you would be using two classes: **Receipt** (for documents) and **ReceiptDetail** (for document lines).

In a similar way, let's create a Production class to be used for manufactured products:

```
CLASS Production 'Production';
date 'Date' = DATA DATE
(Production) IN id;
item 'Product' = DATA Item
(Production);
nameItem 'Product' (Production
p) = name(item(p)) IN id;
posted 'Completed' = DATA BOOLEAN
 (Production);

FORM productions 'Production'
    OBJECTS p = Production
    PROPERTIES(p) date,
nameItem, posted, NEW, DELETE
;
NAVIGATOR {
    NEW productions;
}
```

So far, we've been only creating regular classes without any inheritance. To implement the batch logic, let's create an abstract class called **Batch**:

```
CLASS ABSTRACT Batch 'Batch';
date 'Date' = ABSTRACT DATE (Batch) IN
id;
item 'Product' = ABSTRACT Item (Batch);
nameItem 'Product' (Batch b) = name
(item(b));
type 'Type' = ABSTRACT STRING[30]
(Batch);

FORM batches 'Batches'
    OBJECTS b = Batch
    PROPERTIES(b) READONLY date,
nameItem, type, objectClassName
;
NAVIGATOR {
    NEW batches;
}
```

Each object of this class will correspond to one batch of a particular product. All of its properties will be declared abstract — that is, their implementation will differ depending on the class of a particular batch.

You cannot directly create objects of the abstract **Batch** class in the system. To do that, you need to declare specific classes that will be inherited from it. In particular, let's create a class for batches formed from the receipt of products:

```
CLASS ReceiptBatch 'Arriva
l based batch';
batch (Receipt receipt) =
AGGR ReceiptBatch WHERE
posted(receipt);
```

Use the AGGR operator for each object of the **Receipt**,  class with a defined **posted** property to automatically create (and delete) an object of the **Receipt Batch** class. At this time, the system creates two properties with reciprocal object links: **batch(Receipt r)** and **receipt(ReceiptBatch b)**.

Now we need to inherit the **ReceiptBatch** class from **Batch** to make sure that all batches created by the receipt document also become objects of the abstract class (that is, previously declared batches):

```
EXTEND CLASS ReceiptBatch : Batch;
date(ReceiptBatch rb) += date(receipt(rb));
item(ReceiptBatch rb) += item(receipt(rb));
type(ReceiptBatch rb) += 'Arrival' IF rb IS
 ReceiptBatch;
```

Inheritance is implemented with the help of the EXTEND CLASS operator. After that, for each abstract property of **Batch**, we define how exactly it should be calculated for a specific **ReceiptBatch** class. Date and product values are retrieved from the receipt document through the **receipt(ReceiptBatch b)** link. The necessary string is substituted into the batch type under the condition that the object belongs to the right class (otherwise, the expression will be defined for objects of any class, and the system will generate a signature mismatch error).

Note that you could inherit a class directly while declaring the **ReceiptBatch** class.

In a similar fashion, let's create batches for manufacturing documents:

```
CLASS ProductionBatch 'Production based batch';
batch (Production production) = AGGR
ProductionBatch WHERE posted(production);

EXTEND CLASS ProductionBatch : Batch;
date(ProductionBatch rb) += date(production(rb));
item(ProductionBatch rb) += item(production(rb));
type(ProductionBatch rb) += 'Production' IF rb IS
 ProductionBatch;
```

If necessary, you can create a class for manual batch entry by the user:

```
CLASS UserBatch 'Manually
created batch';
date 'Date' = DATA DATE
(UserBatch) IN id;
item 'Product' = DATA Item
(UserBatch);
nameItem 'Product' (UserBatch
b) = name(item(b));

FORM userBatches 'Batches
(manual)'
    OBJECTS b = UserBatch
    PROPERTIES(b) date,
nameItem, NEW, DELETE
;

NAVIGATOR {
    NEW userBatches;
}

EXTEND CLASS UserBatch : Batch;
date(UserBatch ub) += date(ub);
item(UserBatch ub) += item(ub);
type(UserBatch ub) += 'Manual'
IF ub IS UserBatch;
```