Proposal for Object-Reuse Documentation

The object-reuse section should be moved from the Data Types section and be combined with the Passing Functions to Flink section in the DataSet programming guide.

For Reviewers:

Please check if these rules need to be reflected in code changes, i.e., do we need to update code to make these rules correct.
Please list any required changes (including open JIRA issues below).

TODO:

- Update JavaDocs of InputFormat.nextRecord(reuse) and state rules for reuse object
  - reuse may be modified
  - reuse may be emitted
  - reuse must not be read
  - reuse must not be remembered across function call
-

Object Handling in Functions

Flink's runtime code exchanges data with user functions in form of Java objects. Functions receive input objects from the runtime as method parameters and return output objects as result. Because these objects can be accessed by user functions and runtime code, it is very important to understand and follow the rules about how the user code may access, i.e., read and modify, these objects.

User functions receive objects from the Flink runtime either as regular method parameters (like a MapFunction) or through an Iterable parameter (like a GroupReduceFunction). We refer to objects that the runtime passes to a user function as "*input objects*". User functions can emit objects to the Flink runtime either as a method return value (like a MapFunction) or through a Collector (like a FlatMapFunction). We refer to objects which have been emitted by the user function to the runtime as "*output objects*".

Flink's DataSet API features two modes that differ in how Flink's runtime code reuses and creates new object instances. This behavior affects the guarantees and constraints for how user

functions may interact with objects. The following sections define these rules and give coding guidelines to write safe user function code.

Object-Reuse Disabled (DEFAULT)

By default, Flink operates in object-reuse disabled mode. This mode ensures that functions always receive new input objects within a function call. The object-reuse disabled mode gives better guarantees and is safer to use. However, it comes with a certain processing overhead and might cause higher Java garbage collection activity.

Reading input objects	Within a method call it is guaranteed that the values of input objects do not change. This includes objects served by Iterables. For example it is valid to collect input objects served by an Iterable in a list or map. Note: Objects may be modified after the method call is left. Therefore, it is not safe to remember objects across function calls.
Modifying input objects	You may modify input objects.
Emitting input objects	You may emit input objects. Note, the value of an input object may have changed after it was emitted. It is not safe to read an input object after emission.
Reading output objects	You must treat output objects as modified, i.e., an object which was given to a Collector or returned as method result might have changed its value. It is not safe to read an output object.
Modifying output objects	You may modify an output object and emit it again.

Coding guidelines for object-reuse disabled:

- Do not remember and read input objects across method calls.
- Do not read objects after you emitted them.

Object-Reuse Enabled

In object-reuse mode, Flink's runtime will minimize the number of new object instantiations. This improves the performance and reduces the Java garbage collection pressure. The object-reuse enabled mode is activated by calling ExecutionConfig.enableObjectReuse().

Reading input objects received as regular method parameters	Input objects received as regular method arguments are not modified by the runtime until the method is left. Objects may be modified after method call is left. Hence, it is not safe to remember objects across function calls.

Reading input objects received from an Iterable parameter	Input objects received from an Iterable are only valid until the next() method is called. An Iterable or Iterator may serve the same object instance several times. It is not safe to remember input objects received from an Iterable, e.g., by putting them in a list or map.
Modifying input objects	You must not modify input objects, except for input objects of MapFunction, FlatMapFunction, MapPartitionFunction, GroupReduceFunction, GroupCombineFunction, CoGroupFunction, and InputFormat.next(reuse).
Emitting input objects	You must not emit input object, except for input objects of MapFunction, FlatMapFunction, MapPartitionFunction, GroupReduceFunction, GroupCombineFunction, CoGroupFunction, and InputFormat.next(reuse).
Reading output objects	You must treat output objects as modified, i.e., an object which was given to a Collector or returned as method result might have changed its value. It is not safe to read an output object.
Modifying output objects	You may modify an output object and emit it again.

Coding guidelines for object-reuse enabled:

- Do not modify an input object, except for input objects of MapFunction, FlatMapFunction, MapPartitionFunction, GroupReduceFunction, GroupCombineFunction, CoGroupFunction, and InputFormat.next(reuse).
- Do not remember input objects received from an Iterable.
- Do not remember and read input objects across method calls.
- To reduce object instantiations, you can always emit a dedicated output object which is repeatedly modified but never read.