# Description of Design Problem

* The system must be able to accept other payment methods example cash.
* Currently the transaction controller is creating the coin receiver class explicitly.

# Candidate Design Patterns Considered

A creational design pattern is required.

1. Factory Method.
   * Using an interface to create objects and let subclasses decide which class to instantiate.
2. Singleton
   * Ensure a class has only one instance, and provide a global point of access to it.
3. Prototype
   * A fully initialized instance to be copied or cloned.

# Pattern Chosen

Parameterized Factory design pattern implemented with Singleton.

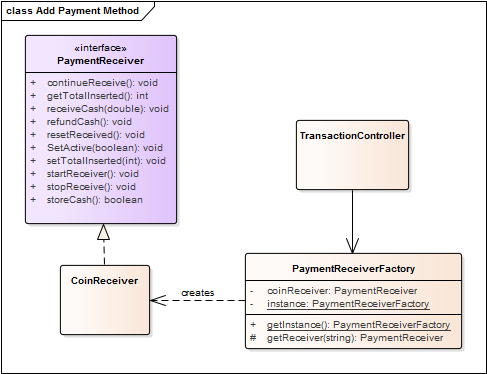
Factory design pattern allow new payment methods to be added easily. Singleton ensure that there is only one instance of the payment method.

# Participants

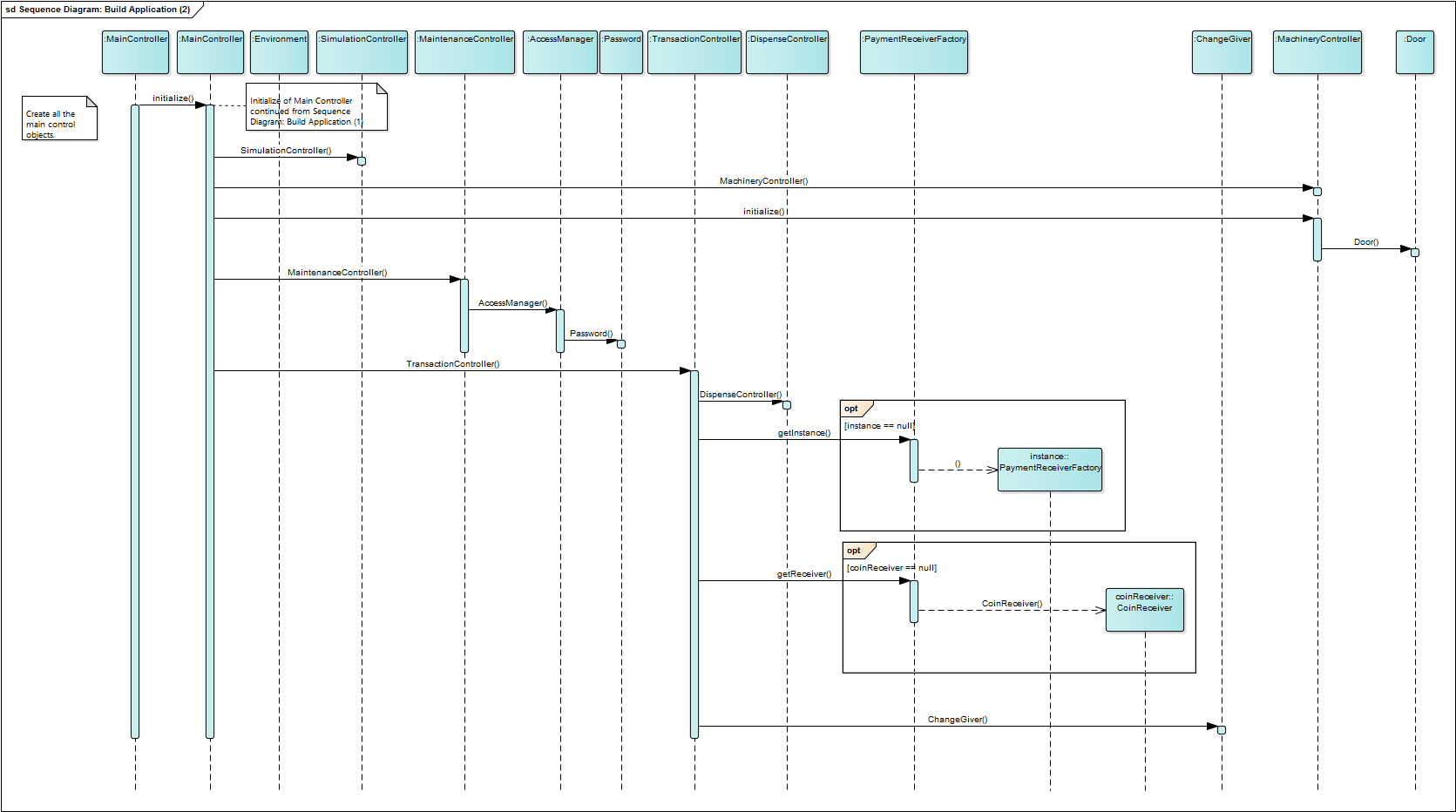
  The classes and objects participating in this pattern are:

* **PaymentReceiver**
  + defines the interface of objects the factory method creates
* **ConcreteReceiver**  **(CashReceiver)**
  + implements the PaymentReceiver interface
* **PaymentReceiverFactory** 
  + declares the factory method, which returns only an instance of an object of type PaymentReceiverFactory.
* **Client (TransactionController)**
  + Initialized the PaymentReceiverFactory

# Class Diagram



# Collaboration Diagram



# Implementation Issues

There are 2 main variations of the Factory Method design pattern. To enforce that only 1 instance of the factory class, a parameterized factory pattern would be most suitable.

The PaymentReceiverFactory will not be subclassing because its main functionality is creating instances of PaymentReceiver. Enforcing singleton ensures that only 1 instance of PaymentReceiverFactory and only 1 instance of coinReceiver is created.