Factory Methods

Factory Method is a method that creates objects.

They are often used instead of constructors, when

(a) creating object is complex

(b) you want to control object creation

(c) you want to create *subclasses of the same type.*

Factory Methods

A *simple factory method* is useful for simplifying object creation.

For example:

```
Calendar cal = Calendar.newInstance( );
```

What it actually did:

```
cal.getClass().getName()
"java.util.GregorianCalendar"
```

Simple Factory Method

Calendar.getInstance() controls object creation so it can create a calendar for your location.

You can't write "new Calendar( )".

```
Calendar cal = Calendar.getInstance( );
Calendar cal =
           Calendar.getInstance( locale );
cal.toString()
   //Thai Locale:    24 Jan 2561
   //U.S. Locale:    Jan 24, 2018
   //Japan Locale:   2018 Jan 24
```

Factory Method can create subtype

A factory method has freedom to create an object of a subclass.

```
Calendar cal = Calendar.getInstance( );
// What did it create?
// Use getClass() to get the class name
System.out.println(
            cal.getClass().getName() );


java.util.GregorianCalendar
```

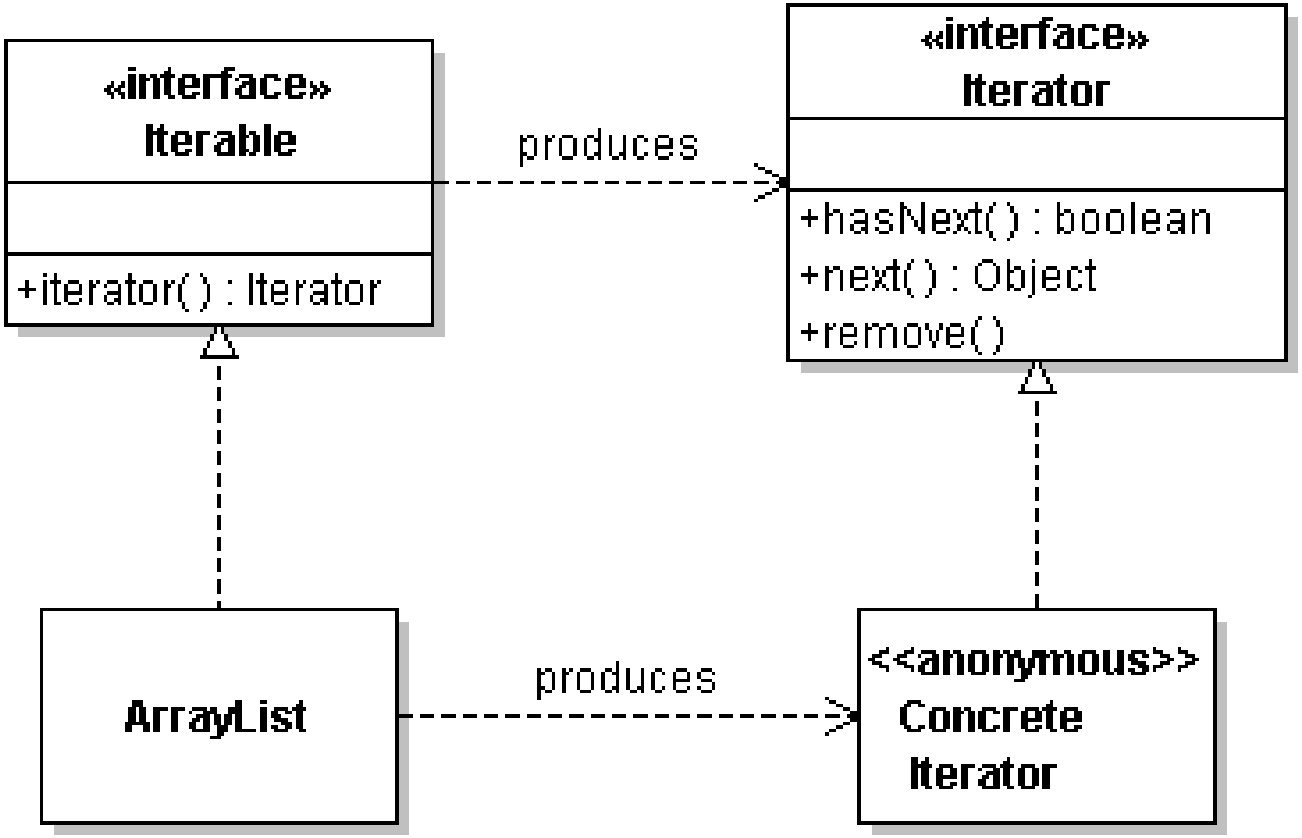
Java API has Many Factory Methods

```
// An immutable date object
LocalDate date = LocalDate.now( );
// A Double object from a String
Double pi = Double.valueOf("3.14159");
// Connection to a database for JDBC
// creating a connection is complex and
// might fail, so use factory method
String url = "jdbc:mysql://somehost/oopdb";
Connection connection =
      DriverManager.getConnection(url);
```

iterable.iterator( ) is Factory Method

How to create iterators?

We don't want the application to know the logic for creating different kinds of iterators.



Creating Money in Purse App

In the PurseConsole, we have some code like this:

```
private Money makeMoney(double value) {
  if (value==1.0 || value==2.0 || value=5.0 || value=10.0)
      return new Coin(value,"Baht");
  if (value==20 || value==50 || value=100 || value==500)
      return new BankNote(value, "Baht");
  if (value==1000)
      return new BankNote(value, "Baht");
  return null; // or throw IllegalArgumentException
}
```

Any Problem?

Requirements Change

Localize Change

Design 1 class to be responsible for creating all money.

The Purse app always uses this class to create Money.

If Money changes, we only need to modify one class.

```
MoneyFactory factory =
                MoneyFactory.getInstance( );

Money m = new Money(10.0);
```

Need only 1 factory object

We only need 1 MoneyFactory object.

So, we would like getInstance() to always return the *same* object.

```
MoneyFactory factory1 =
                MoneyFactory.getInstance( );

MoneyFactory factory2 =
                MoneyFactory.getInstance( );

System.out.println(factory1 == factory2);
true
```