Understand the Java Supplier
Functional Interface

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand foundational functional programming features in Java, e.g.,
  - Lambda expressions
  - Method & constructor references
- Key functional interfaces
  - Predicate
  - Function
  - BiFunction
  - Supplier

```java
public interface Supplier<T>

Type Parameters:
T - the type of results supplied by this supplier

Functional Interface:
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

@FunctionalInterface
class Supplier<T>

public interface Supplier<T>

Represents a supplier of results.

There is no requirement that a new or distinct result be returned each time the supplier is invoked.

This is a functional interface whose functional method is get().
```
Learning Objectives in this Part of the Lesson

- Understand foundational functional programming features in Java
- Learn how to apply Java suppliers in concise example programs

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8](http://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8)
Learning Objectives in this Part of the Lesson

- Understand foundational functional programming features in Java
- Learn how to apply Java suppliers in concise example programs
- The examples showcase the Java collections framework

See docs.oracle.com/javase/8/docs/technotes/guides/collections
Overview of the Supplier Functional Interface
Overview of Supplier Functional Interface

- A Supplier returns a value & takes no parameters, e.g.,
  - public interface Supplier<T> { T get(); }

See docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html
Overview of Supplier Functional Interface

- A `Supplier` returns a value & takes no parameters, e.g.,
  - `public interface Supplier<T> { T get(); }`

  *Supplier is a generic interface that is parameterized by one reference type*
Overview of Supplier Functional Interface

- A *Supplier* returns a value & takes no parameters, e.g.,
  - public interface Supplier<T> { T get(); }

*Its single abstract method is passed no parameters & returns a value of type T.*
Overview of Supplier Functional Interface

- A Supplier returns a value & takes no parameters, e.g.,
  ```java
  public interface Supplier<T> { T get(); }
  ```

Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of 
  + being + " = "
  + disposition.orElseGet(() -> "unknown")");

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex6
Overview of Supplier Functional Interface

- A Supplier returns a value & takes no parameters, e.g.,
  - public interface Supplier<T> { T get(); }

```
Map<String, String> beingMap = new HashMap<String, String>() {
    { put("Demon", "Naughty"); put("Angel", "Nice"); } }

String being = ...;

Optional<String> disposition = Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of " + being + " = " +
                   disposition.orElseGet(() -> "unknown"));
```

Create a map associating each being with its personality traits
Overview of Supplier Functional Interface

• A Supplier returns a value & takes no parameters, e.g.,
  • public interface Supplier<T> { T get(); }

Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } }

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "+ being + " = "+ disposition.orElseGet(() -> "unknown");

Get the name of a being from somewhere (e.g., prompt user)
A `Supplier` returns a value & takes no parameters, e.g.,

```java
public interface Supplier<T> { T get(); }
```

```java
Map<String, String> beingMap = new HashMap<String, String>() {
    { put("Demon", "Naughty"); put("Angel", "Nice"); }
};
```

```java
String being = ...;
```

```java
Optional<String> disposition = Optional.ofNullable(beeingMap.get(being));
```

```java
System.out.println("disposition of 
    + being + " = "
    + disposition.orElseGet(() -> "unknown"));
```

See [docs.oracle.com/javase/8/docs/api/java/util.Optional.html#ofNullable](https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html#ofNullable)
Overview of Supplier Functional Interface

- A Supplier returns a value & takes no parameters, e.g.,
  - public interface Supplier<T> { T get(); }

Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } }; 

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "+ being + " = "+
  disposition.orElseGet(() -> "unknown");

See docs.oracle.com/javase/8/docs/api/java/util/Optional.html
Overview of Supplier Functional Interface

- A Supplier returns a value & takes no parameters, e.g.,
  
  public interface Supplier<T> { T get(); }

  Map<String, String> beingMap = new HashMap<String, String>()
      { { put("Demon", "Naughty"); put("Angel", "Nice"); } };

  String being = ...;

  Optional<String> disposition =
      Optional.ofNullable(beingMap.get(being));

  System.out.println("disposition of "
      + being + " = "
      + disposition.orElseGet(() -> "unknown"));

  Returns value if being is non-null

See docs.oracle.com/javase/8/docs/api/java/util/Optional.html#orElseGet
Overview of Supplier Functional Interface

- A *Supplier* returns a value & takes no parameters, e.g.,
  - `public interface Supplier<T> { T get(); }`

```java
Map<String, String> beingMap = new HashMap<String, String>() {
    { put("Demon", "Naughty"); put("Angel", "Nice"); }
};

String being = ...;

Optional<String> disposition =
    Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "
    + being + " = "
    + disposition.orElseGet(() -> "unknown"));
```

Returns supplier lambda value if being is not found
Overview of Supplier Functional Interface

- A Supplier returns a value & takes no parameters, e.g.,
  - public interface Supplier<T> { T get(); }

Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition = Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of " + being + " = " + disposition.orElse("unknown"));

Could also use orElse()

See docs.oracle.com/javase/8/docs/api/java/util/Optional.html#orElse
Overview of Supplier Functional Interface

• A Supplier returns a value & takes no parameters, e.g.,
  • public interface Supplier<T> { T get(); }

class Optional<T> {
  ...    
  public T orElseGet(Supplier<? extends T> other) {
    return value != null
    ? value
    : other.get();
  }
}

Here’s how the orElseGet() method uses the supplier passed to it
Overview of Supplier Functional Interface

- A Supplier returns a value & takes no parameters, e.g.,
  - public interface Supplier<T> { T get(); }

```java
class Optional<T> {
    ...
    public T orElseGet(Supplier<? extends T> other) {
        return value != null
            ? value
            : other.get();
    }
}
```

The string literal "unknown" is bound to the supplier lambda parameter
Overview of Supplier Functional Interface

- A **Supplier** returns a value & takes no parameters, e.g.,
- public interface Supplier<T> { T get(); }

```java
class Optional<T> {
  ...
  public T orElseGet(Supplier<? extends T> other) {
    return value != null
      ? value
      : other.get();
  }
}
```

The string "unknown" is returned by orElseGet() if the value is null.
Another Example of the Supplier Interface
A Supplier can also be used for a zero-param constructor reference e.g.,

```java
public interface Supplier<T> { T get(); }

class CrDemo implements Runnable {
    String mString;

    void zeroParamConstructorRef() {
        Supplier<CrDemo> factory = CrDemo::new;
        CrDemo crDemo = factory.get();
        crDemo.run();
    }

    void run() { System.out.println(mString); }
    ...
}
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex7](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex7)
Another Example of the Supplier Interface

- A Supplier can also be used for a zero-param constructor reference e.g.,

```java
public interface Supplier<T> { T get(); }
```

```java
class CrDemo implements Runnable {
    String mString;

    void zeroParamConstructorRef() {
        Supplier<CrDemo> factory = CrDemo::new;
        CrDemo crDemo = factory.get();
        crDemo.run();
    }

    @Override
    void run() {
        System.out.println(mString);
    }
...
}
```

Create a supplier that’s initialized with a zero-param constructor reference for CrDemo

Another Example of the Supplier Interface

- A `Supplier` can also be used for a zero-param constructor reference e.g.,
  ```java
  public interface Supplier<T> { T get(); }
  class CrDemo implements Runnable {
      String mString;

      void zeroParamConstructorRef() {
          Supplier<CrDemo> factory = CrDemo::new;
          CrDemo crDemo = factory.get();
          crDemo.run();
      }

      @Override
      void run() { System.out.println(mString); }
  }
  ...
  ```

- `get()` creates a `CrDemo` object using a constructor reference for the `CrDemo` "default" constructor
Another Example of the Supplier Interface

- A Supplier can also be used for a zero-param constructor reference e.g.,
  
  ```java
  public interface Supplier<T> { T get(); }
  
  class CrDemo implements Runnable {
      String mString;

      void zeroParamConstructorRef() {
          Supplier<CrDemo> factory = CrDemo::new;
          CrDemo crDemo = factory.get();
          crDemo.run();
      }
  }

  @Override
  void run() { System.out.println(mString); }
  ...
Another Example of the Supplier Interface

- Constructor references simplify creation of parameterizable factory methods
  - public interface Supplier<T> { T get(); }

```java
class CrDemo implements Runnable {
    ...
    static class CrDemoEx extends CrDemo {
        @Override
        public void run() {
            System.out.println(mString.toUpperCase());
        }
    }
    ...
}
```

This class extends CrDemo & overrides its run() method to uppercase the string.
Another Example of the Supplier Interface

- Constructor references simplify creation of parameterizable factory methods

```java
public interface Supplier<T> { T get(); }

class CrDemo implements Runnable {
    ...
    static class CrDemoEx
        extends CrDemo {

            @Override
            public void run() {
                System.out.println(mString.toUpperCase());
            }
        }
    ...

Print the upper-cased value of mString
Another Example of the Supplier Interface

- Constructor references simplify creation of parameterizable factory methods
  - public interface Supplier<T> { T get(); }
  - class CrDemo implements Runnable {
      ...
      void zeroParamConstructorRefEx() {
        Supplier<CrDemo> crDemoFactory = CrDemo::new;
        Supplier<CrDemoEx> crDemoFactoryEx = CrDemoEx::new;

        runDemo(crDemoFactory);
        runDemo(crDemoFactoryEx);
      }
      ...
  }

Demonstrate how suppliers can be used as factories for multiple zero-parameter constructor references
Another Example of the Supplier Interface

- Constructor references simplify creation of parameterizable factory methods
  - public interface Supplier<T> { T get(); }
  - class CrDemo implements Runnable {
      ...
      void zeroParamConstructorRefEx() {
          Supplier<CrDemo> crDemoFactory = CrDemo::new;
          Supplier<CrDemoEx> crDemoFactoryEx = CrDemoEx::new;

          runDemo(crDemoFactory);
          runDemo(crDemoFactoryEx);
      }
      ...
  }
Another Example of the Supplier Interface

- Constructor references simplify creation of parameterizable factory methods

  ```
  public interface Supplier<T> { T get(); }
  class CrDemo implements Runnable {
      ...
      void zeroParamConstructorRefEx() {
          Supplier<CrDemo> crDemoFactory = CrDemo::new;
          Supplier<CrDemoEx> crDemoFactoryEx = CrDemo::new;

          runDemo(crDemoFactory);
          runDemo(crDemoFactoryEx);
      }
  }
  ```

This helper method invokes the given supplier to create a new object & call its run() method.
Another Example of the Supplier Interface

- Constructor references simplify creation of parameterizable factory methods
- public interface Supplier<T> { T get(); }
  class CrDemo implements Runnable {
    ...
    <T> extends Runnable> void runDemo(Supplier<T> factory) {
      factory.get().run();
    }
    ...

Use the given factory to create a new object & call its run() method
Another Example of the Supplier Interface

- Constructor references simplify creation of parameterizable factory methods
- `public interface Supplier<T> { T get(); }`
- `class CrDemo implements Runnable {
  ...
  <T> extends Runnable> void runDemo(Supplier<T> factory) {
      factory.get().run();
  }
  ...
}

This call encapsulates details of the concrete constructor that’s used to create an object!
• Arbitrary constructors w/params can also be supported in Java, e.g.

• public interface Supplier<T> { T get(); }  
  class CrDemo implements Runnable { ...  
    interface TriFactory<A, B, C, R> { R of(A a, B b, C c); }  

**Custom functional interfaces can be defined for arbitrary constructors w/params**

    void threeParamConstructorRef() {  
      TriFactory<String, Integer, Long, CrDemo> factory =  
        CrDemo::new;  

      factory.of("The answer is ", 4, 2L).run();  
    }

    CrDemo(String s, Integer i, Long l)  
    { mString = s + i + l; } ...
Another Example of the Supplier Interface

- Arbitrary constructors w/params can also be supported in Java, e.g.
  
  ```java
  public interface Supplier<T> { T get(); }
  class CrDemo implements Runnable { ...
      interface TriFactory<A, B, C, R> { R of(A a, B b, C c); }

      void threeParamConstructorRef() {
          TriFactory<String, Integer, Long, CrDemo> factory =
              CrDemo::new;

          factory.of("The answer is ", 4, 2L).run();
      }

      CrDemo(String s, Integer i, Long l) {
          mString = s + i + l; }
  }
  ```

*This factory method creates an instance of R using params a, b, & c*
Another Example of the Supplier Interface

- Arbitrary constructors w/params can also be supported in Java, e.g.
  - public interface Supplier<T> { T get(); }
  - class CrDemo implements Runnable {
    
      interface TriFactory<A, B, C, R> { R of(A a, B b, C c); }
    
    void threeParamConstructorRef() {
      TriFactory<String, Integer, Long, CrDemo> factory =
        CrDemo::new;

      factory.of("The answer is ", 4, 2L).run();
    }
  }

  Create a factory that’s initialized with a three-param constructor reference

  CrDemo(String s, Integer i, Long l)
  { mString = s + i + l; } ...
Another Example of the Supplier Interface

- Arbitrary constructors w/params can also be supported in Java, e.g.
  - public interface Supplier<T> { T get(); }
  - class CrDemo implements Runnable { ...  
    - interface TriFactory<A, B, C, R> { R of(A a, B b, C c); }

    void threeParamConstructorRef() {
      TriFactory<String, Integer, Long, CrDemo> factory = CrDemo::new;

      factory.of("The answer is ", 4, 2L).run();
    }

    CrDemo(String s, Integer i, Long l)  
    { mString = s + i + l; } ...
End of the Java Supplier Functional Interface