

# The Java Supplier Functional Interface: Constructor References

Douglas C. Schmidt

[d.schmidt@vanderbilt.edu](mailto:d.schmidt@vanderbilt.edu)

[www.dre.vanderbilt.edu/~schmidt](http://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

---

- Know how to apply Java Supplier to another concise example that shows how to use Supplier objects to hold constructor references & create objects dynamically



```
class CrDemo implements Runnable {  
    String mString;  
  
    void zeroParamConstructorRef() {  
        Supplier<CrDemo> factory =  
            CrDemo::new;  
        CrDemo crDemo = factory.get();  
        crDemo.run();  
    }  
}
```

---

# Applying the Supplier Interface for Default Constructors

# Applying the Supplier Interface for Default Constructors

---

- A *Supplier* is often used for 0-param (default) constructor references, e.g.

```
class CrDemo implements Runnable {  
    String mString;  
  
    CrDemo() {  
        mString = "hello";  
    }  
  
    ...  
}
```

# Applying the Supplier Interface for Default Constructors

---

- A *Supplier* is often used for 0-param (default) constructor references, e.g.

```
class CrDemo implements Runnable {  
    String mString;  
  
    CrDemo() {  
        mString = "hello";  
    }  
  
    ...  
}
```

# Applying the Supplier Interface for Default Constructors

- A *Supplier* is often used for 0-param (default) constructor references, e.g.

```
class CrDemo implements Runnable {  
    String mString;  
  
    CrDemo () {  
        mString = "hello";  
    }  
    ...  
}
```

*The 0-param (default) constructor initializes a String object to "hello"*

# Applying the Supplier Interface for Default Constructors

- A *Supplier* is often used for 0-param (default) constructor references, e.g.

```
class CrDemo implements Runnable {  
  
    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*

# Applying the Supplier Interface for Default Constructors

- A *Supplier* is often used for 0-param (default) constructor references, e.g.

```
class CrDemo implements Runnable {  
  
    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

# Applying the Supplier Interface for Default Constructors

- A *Supplier* is often used for 0-param (default) constructor references, e.g.

```
class CrDemo implements Runnable {  
  
    void zeroParamConstructorRef() {  
        Supplier<CrDemo> factory = CrDemo::new;  
        CrDemo crDemo = factory.get();  
        crDemo.run();  
    }  
  
    @Override  
    void run() { System.out.println(mString); }  
    ...  
}
```

*Call the run() hook method in CrDemo to print the String value*

---

# Simplifying Factory Methods with Constructor References

# Simplifying Factory Methods with Constructor References

- Constructor references can simplify the creation of factory methods

```
class CrDemo implements Runnable {
```

```
...
```

```
static class CrDemoEx extends CrDemo {
```

*This class extends CrDemo & overrides its run() hook method to uppercase the string*

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

# Simplifying Factory Methods with Constructor References

- Constructor references can simplify the creation of factory methods, e.g.

```
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*

# Simplifying Factory Methods with Constructor References

- Constructor references can simplify the creation of factory methods, e.g.

```
class CrDemo implements Runnable {  
    ...  
    void zeroParamConstructorRefEx() {
```

*Demonstrate how suppliers can be used as factories  
for multiple zero-parameter constructor references*

```
Supplier<CrDemo> crDemoFactory = CrDemo::new;  
Supplier<CrDemoEx> crDemoFactoryEx = CrDemoEx::new;  
  
runDemo(crDemoFactory);  
runDemo(crDemoFactoryEx);  
}  
...
```

# Simplifying Factory Methods with Constructor References

- Constructor references can simplify the creation of factory methods, e.g.

```
class CrDemo implements Runnable {  
    ...  
    void zeroParamConstructorRefEx() {
```

*Assign a constructor reference to a supplier that acts as a factory for a zero-param object of CrDemo/CrDemoEx*

```
Supplier<CrDemo> crDemoFactory = CrDemo::new;  
Supplier<CrDemoEx> crDemoFactoryEx = CrDemoEx::new;  
  
runDemo(crDemoFactory);  
runDemo(crDemoFactoryEx);  
}  
...
```

# Simplifying Factory Methods with Constructor References

- Constructor references can simplify the creation of factory methods, e.g.

```
class CrDemo implements Runnable {  
    ...  
    void zeroParamConstructorRefEx() {
```

```
        Supplier<CrDemo> crDemoFactory = CrDemo::new;  
        Supplier<CrDemoEx> crDemoFactoryEx = CrDemoEx::new;
```

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

*This helper method invokes the given supplier to create a new object & call its run() method*

# Simplifying Factory Methods with Constructor References

- Constructor references can simplify the creation of factory methods, e.g.

```
class CrDemo implements Runnable {  
    ...  
    static <T extends Runnable> void runDemo(Supplier<T> factory) {  
        factory.get().run();  
    }  
    ...  
}
```

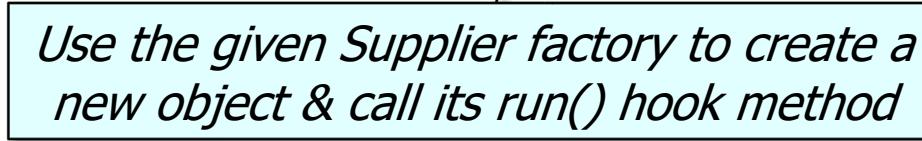


*runDemo() is parameterized by a type  
that extends the Runnable interface*

# Simplifying Factory Methods with Constructor References

- Constructor references can simplify the creation of factory methods, e.g.

```
class CrDemo implements Runnable {  
    ...  
    static <T extends Runnable> void runDemo(Supplier<T> factory) {  
        factory.get().run();  
    }  
    ...  
}
```

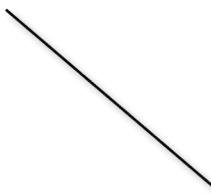


*Use the given Supplier factory to create a new object & call its run() hook method*

# Simplifying Factory Methods with Constructor References

- Constructor references can simplify the creation of factory methods, e.g.

```
class CrDemo implements Runnable {  
    ...  
    static <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!*

# Simplifying Factory Methods with Constructor References

- Constructor references can simplify the creation of factory methods, e.g.

```
class CrDemo implements Runnable {  
    ...  
    static <T extends Runnable> void runDemo(Supplier<T> factory) {  
        factory.get().run();  
    }  
    ...
```



*Call the appropriate run() hook  
method to print the String*

---

# References to Constructors with Arbitrary Parameters

# References to Constructors with Arbitrary Parameters

- References to constructors w/arbitrary params are supported in modern Java

```
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; } ...
```

This capability is unrelated to the Supplier interface..

# References to Constructors with Arbitrary Parameters

- References to constructors w/arbitrary params are supported in modern Java

```
class CrDemo implements Runnable { ...  
    interface TriFactory<A, B, C, R> { R of(A a, B b, C c); }
```

*This factory method creates an instance of R using params a, b, & 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; } ...
```

# References to Constructors with Arbitrary Parameters

- References to constructors w/arbitrary params are supported in modern Java

```
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; } ...
```

# References to Constructors with Arbitrary Parameters

- References to constructors w/arbitrary params are supported in modern Java

```
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; } ...
```

*Create/print a 3-param  
instance of CrDemo*

---

# End of the Java Supplier Functional Interface: Constructor References