Mobiliser Platform – Foundation

Introduction to the Spring Framework

Spring Framework

What it is and Why it is Needed?

Spring Framework – High-level Description

The Spring Framework is open source software released under the Apache 2.0 license that provides both a design time framework for Java application development and a runtime container for Java object management.

Spring Framework – High-level Description

The Spring Framework is open source software released under the Apache 2.0 license that provides both a design time framework for Java application development and a runtime container for Java object management.

The Spring Framework is modular and provides many different types of functionality including:

- Model-View-Controller
- Data Access (provides JDBC and NoSQL interfaces)
- Remote Access Framework (supporting RMI, Corba and HTTP-based protocols such as SOAP)

Spring Framework – High-level Description

The Spring Framework is open source software released under the Apache 2.0 license that provides both a design time framework for Java application development and a runtime container for Java object management.

The Spring Framework is modular and provides many different types of functionality including:

- Model-View-Controller
- Data Access (provides JDBC and NoSQL interfaces)
- Remote Access Framework (supporting RMI, Corba and HTTP-based protocols such as SOAP)

However, the two most important modules offer:

- Inversion of Control (through Dependency Injection)
- Aspect Oriented Programming

Spring Framework – High-level Description

The Spring Framework is open source software released under the Apache 2.0 license that provides both a design time framework for Java application development and a runtime container for Java object management.

The Spring Framework is modular and provides many different types of functionality including:

- Model-View-Controller
- Data Access (provides JDBC and NoSQL interfaces)
- Remote Access Framework (supporting RMI, Corba and HTTP-based protocols such as SOAP)

However, the two most important modules offer:

- Inversion of Control (through Dependency Injection)
- Aspect Oriented Programming

Whilst neither of the following are exclusive use cases, the Spring Framework is most often used:

- For Java application development on the Java Enterprise Edition platform (Java EE).
- As an alternative to the Enterprise Java Bean (EJB) model

Spring Framework

Managing Java Objects

Spring Framework: Object Definition 1/2

The Spring Framework provides an environment for Java object management.

Spring Framework: Object Definition 1/2

The Spring Framework provides an environment for Java object management.

Here is a simple interface that defines a basic Shape object.

```
// A generic interface for all shapes
public interface Shape {
  public double getArea();
}
```

Spring Framework: Object Definition 1/2

The Spring Framework provides an environment for Java object management.

Here is a simple interface that defines a basic `Shape` object.

```
// A generic interface for all shapes
public interface Shape {
  public double getArea();
}
```

The POJO `Rectangle` implements the `Shape` interface

```
// A specific Rectangle object that implement the Shape interface
public class Rectangle implements Shape {
  private double length, width;

  // Constructors – Spring needs a zero argument constructor!
  public Rectangle() {}
  public Rectangle(double length, double width) {
    setLength(length);
    setWidth(width);
  }

 // Implement getter and setter methods
  public double getLength() { return length; }
  public double getWidth()  { return width; }
  public void setLength(double length) { this.length = length; }
  public void setWidth(double width)   { this.width  = width; }

  // Implement interface method
  public double getArea() { return length * width; }
}
```

Spring Framework: Object Definition 2/2

The Spring Framework provides an environment for Java object management.

Here is a simple interface that defines a basic Shape object.

```java
// A generic interface for all shapes
public interface Shape {
  public double getArea();
}
```

The POJO Circle also implements the Shape interface

```java
// A specific Circle object that implement the Shape interface
public class Circle implements Shape {
  private double radius;

  // Constructors - Spring needs a zero argument constructor!
  public Circle() {}
  public Circle(double radius) {
    setRadius(radius);
  }

 // Implement getter and setter methods
  public double getRadius() { return radius; }
  public void setRadius(double radius) { this.radius = radius; }

  // Implement interface method
  public double getArea() { return Math.PI * radius * radius; }
}
```

Spring Framework: Object Declaration

The definitions of the `Rectangle` and `Circle` objects are declared to the Spring Framework in a bean definition file call `applicationContext.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans”
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="shape1" class="Rectangle">
        <property name="length" value="10"/>
        <property name="width" value="20"/>
    </bean>

    <bean id="shape2" class=”Circle">
        <constructor-arg value="10"/>
    </bean>
</beans>
```

Spring Framework: Object Declaration

The definitions of the `Rectangle` and `Circle` objects are declared to the Spring Framework in a bean definition file call `applicationContext.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans”
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="shape1" class="Rectangle">
        <property name="length" value="10"/>
        <property name="width" value="20"/>
    </bean>

    <bean id="shape2" class=”Circle">
        <constructor-arg value="10"/>
    </bean>
</beans>
```

The `Rectangle` class will be identified as `shape1` and instantiated using the zero-argument constructor.
The `length` and `width` properties are set by calling the setter methods

Spring Framework: Object Declaration

The definitions of the `Rectangle` and `Circle` objects are declared to the Spring Framework in a bean definition file call `applicationContext.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans”
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="shape1" class="Rectangle">
        <property name="length" value="10"/>
        <property name="width" value="20"/>
    </bean>

    <bean id="shape2" class=”Circle">
        <constructor-arg value="10"/>
    </bean>
</beans>
```

The `Circle` class will be identified as shape2 and instantiated by passing the radius to the single argument constructor.

Spring Framework: Object Management

The `Rectangle` and `Circle` objects are then instantiated by the Spring Framework using the following coding:

```
import org.springframework.context.*;
import org.springframework.context.support.*;

public class ShapeTest {
  public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("/applicationContext.xml");

    Shape rect = (Shape)context.getBean("shape1");
    Shape circ = (Shape)context.getBean("shape2");

    printInfo(rect);
    printInfo(circ);
  }

  private static void printInfo(Shape shape) {
    System.out.printf("Area of %s is %.2f%n", shape.getClass().getSimpleName(), shape.getArea());
  }
}
```

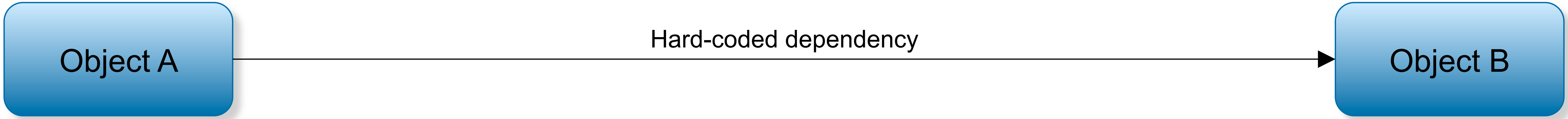
The `Rectangle` and `Circle` classes are instantiated using the identifiers declared in `applicationContext.xml`

Spring Framework

Dependency Injection and Inversion of Control

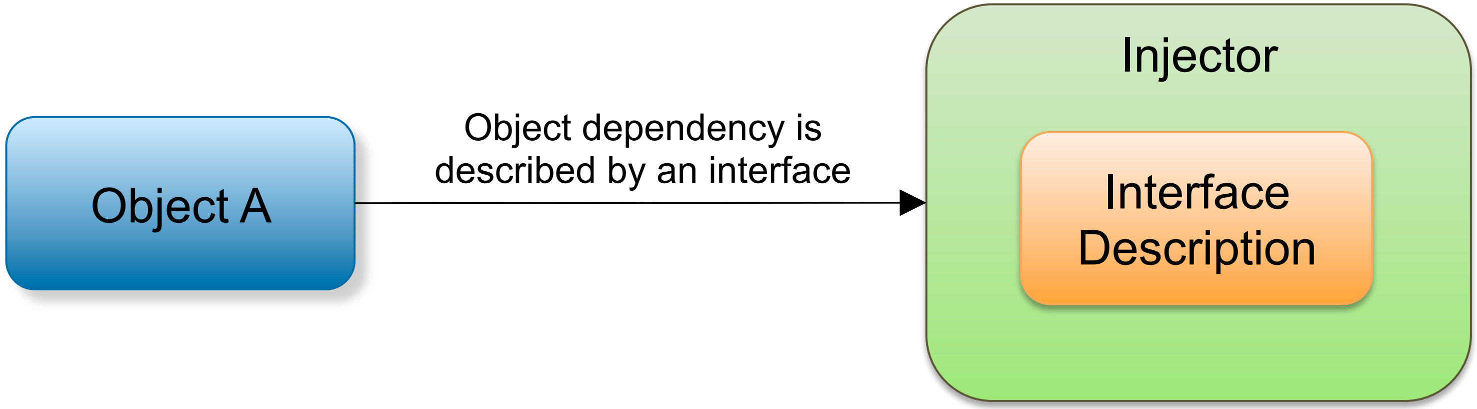
Dependency Injection

Dependency Injection is a software design pattern that allows you to remove hard-coded object dependencies from your application design.



Dependency Injection

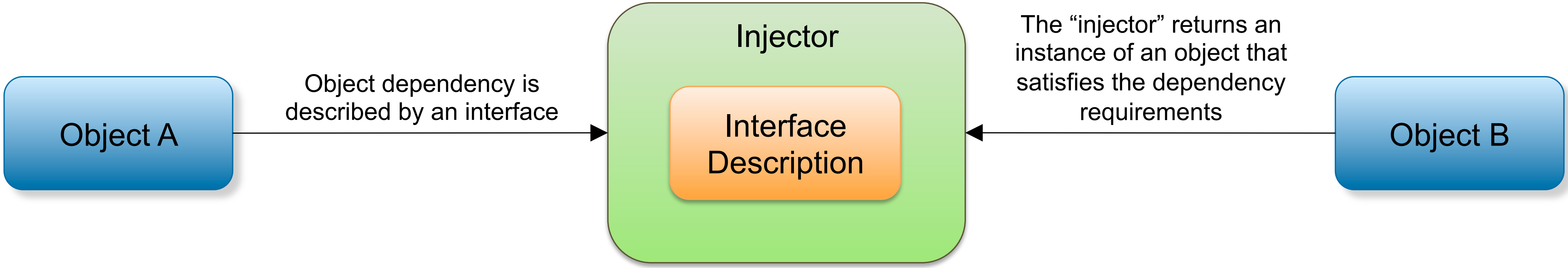
Dependency Injection is a software design pattern that allows you to remove hard-coded object dependencies from your application design.



Object A is written such that it does not specifically depend on Object B, but on ***any*** object that implements the required interface – of which Object B is one of possibly several examples.

Dependency Injection

Dependency Injection is a software design pattern that allows you to remove hard-coded object dependencies from your application design.


Object A
Object dependency is described by an interface
Injector
Interface Description
The “injector” returns an instance of an object that satisfies the dependency requirements
Object B

Object A is written such that it does not specifically depend on Object B, but on ***any*** object that implements the required interface – of which Object B is one of possibly several examples.

Software frameworks that support dependency injection will provide an “Injector”.

The “Injector” performs dependency resolution and returns an instance of the correct object.

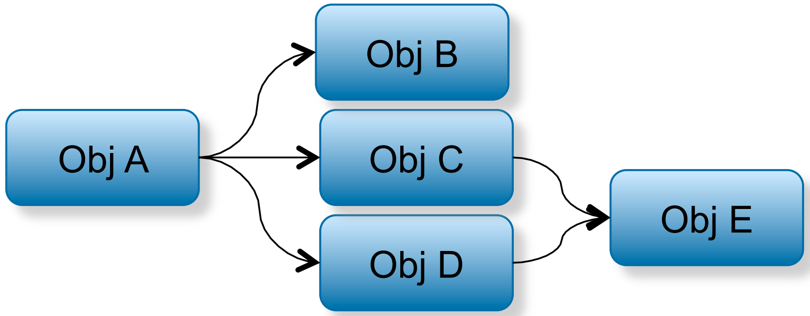
Inversion of Control (or IoC)

Inversion of Control is a programming technique that “inverts” object dependency resolution.

Inversion of Control (or IoC)

Inversion of Control is a programming technique that “inverts” object dependency resolution.

In traditional programming, object dependencies are defined at design time by hard-coding static references into the source code. This technique is both fragile and inflexible.



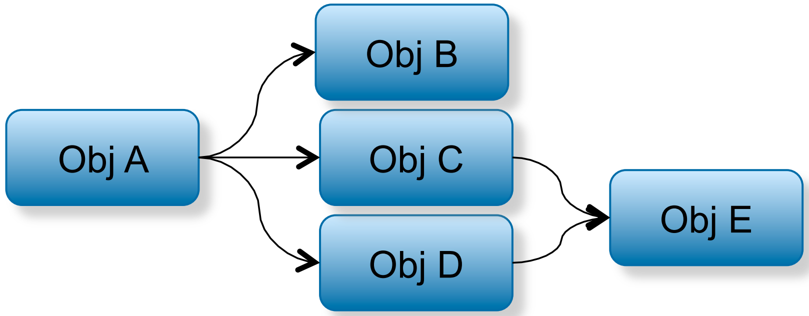
Hard-coded references in source code

Inversion of Control (or IoC)

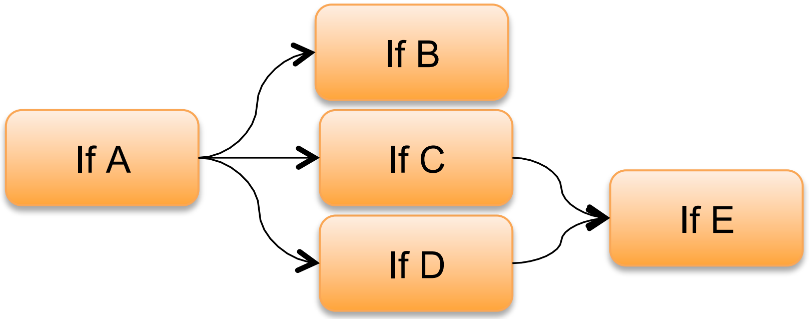
Inversion of Control is a programming technique that “inverts” object dependency resolution.

In traditional programming, object dependencies are defined at design time by hard-coding static references into the source code. This technique is both fragile and inflexible.

However, if for your entire application, a complete graph of object dependencies is first constructed, then at runtime this graph forms the basis upon which object dependencies can be resolved.



Hard-coded references in source code



Object graph describes interfaces and dependencies

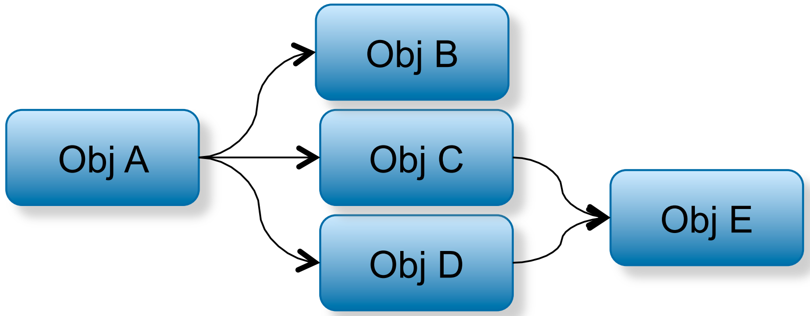
Inversion of Control (or IoC)

Inversion of Control is a programming technique that "inverts" object dependency resolution.

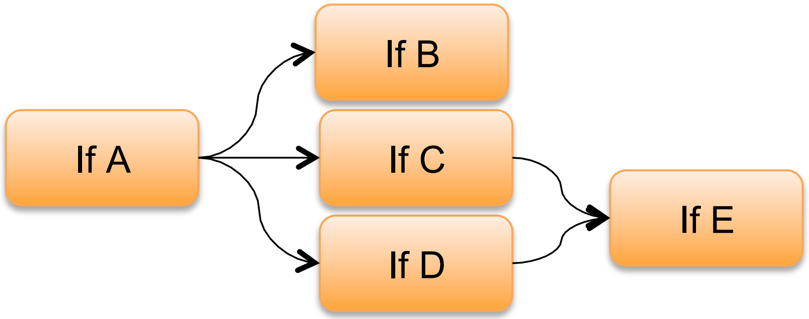
In traditional programming, object dependencies are defined at design time by hard-coding static references into the source code. This technique is both fragile and inflexible.

However, if for your entire application, a complete graph of object dependencies is first constructed, then at runtime this graph forms the basis upon which object dependencies can be resolved.

You are now said to have "inverted" the control and resolution of object dependencies, since using this dependency graph, any object that satisfies the stated interface requirements could be supplied as a runtime candidate.



Hard-coded references in source code



Object graph describes interfaces and dependencies

Inversion of Control through Dependency Injection

Dependency Injection is the design pattern that abstracts the low level dependency between two units of code.

Inversion of Control is a programming paradigm that applies the Dependency Injection pattern to all the units of code in some large-scale unit such as an entire application.

Why is Inversion of Control Necessary?

There are several reason why the ***Inversion of Control*** programming technique is useful:

Why is Inversion of Control Necessary?

There are several reason why the ***Inversion of Control*** programming technique is useful:

1. **Decoupling Execution from Implementation**

   The design-time implementation of an application is performed using functionally specific units of code whose relationship is defined only by an interface contract.

   Which units of code are actually executed at runtime is determined dynamically, based upon nothing more than whether or not that unit of code satisfies a particular interface contract.

Why is Inversion of Control Necessary?

There are several reason why the ***Inversion of Control*** programming technique is useful:

1. **Decoupling Execution from Implementation**

   The design-time implementation of an application is performed using functionally specific units of code whose relationship is defined only by an interface contract.

   Which units of code are actually executed at runtime is determined dynamically, based upon nothing more than whether or not that unit of code satisfies a particular interface contract.

2. **Modular Specificity**

   Each unit of code (or module) can be written to perform one and only one task. This both simplifies development and improves code reuse.

Why is Inversion of Control Necessary?

There are several reason why the ***Inversion of Control*** programming technique is useful:

1. **Decoupling Execution from Implementation**

   The design-time implementation of an application is performed using functionally specific units of code whose relationship is defined only by an interface contract.

   Which units of code are actually executed at runtime is determined dynamically, based upon nothing more than whether or not that unit of code satisfies a particular interface contract.

2. **Modular Specificity**

   Each unit of code (or module) can be written to perform one and only one task. This both simplifies development and improves code reuse.

3. **Interface-only based Interaction**

   No two modules are allowed to interact with each other in any way other than that described in their interface contract.

Why is Inversion of Control Necessary?

There are several reason why the ***Inversion of Control*** programming technique is useful:

1. **Decoupling Execution from Implementation**

   The design-time implementation of an application is performed using functionally specific units of code whose relationship is defined only by an interface contract.

   Which units of code are actually executed at runtime is determined dynamically, based upon nothing more than whether or not that unit of code satisfies a particular interface contract.

2. **Modular Specificity**

   Each unit of code (or module) can be written to perform one and only one task. This both simplifies development and improves code reuse.

3. **Interface-only based Interaction**

   No two modules are allowed to interact with each other in any way other than that described in their interface contract.

4. **Dynamic Module Replacement**

   If two modules both satisfy the same interface contract, then no side-effects are created by swapping one module for another at runtime. The dependent module is unaffected by such changes.

So What Does the Spring Framework Do For Me?

At its most fundamental level, the Spring Framework can be viewed as:

So What Does the Spring Framework Do For Me?

At its most fundamental level, the Spring Framework can be viewed as:

- An ***object factory*** that implements…

So What Does the Spring Framework Do For Me?

At its most fundamental level, the Spring Framework can be viewed as:

- An ***object factory*** that implements…
- ***Inversion of Control*** to manage the instantiation, configuration, decoration and assembly of business objects…

So What Does the Spring Framework Do For Me?

At its most fundamental level, the Spring Framework can be viewed as:

- An ***object factory*** that implements…
- ***Inversion of Control*** to manage the instantiation, configuration, decoration and assembly of business objects…
- By means of the ***Dependency Injection*** pattern.

Spring Framework

Aspect Oriented Programming

Aspect Oriented Programming

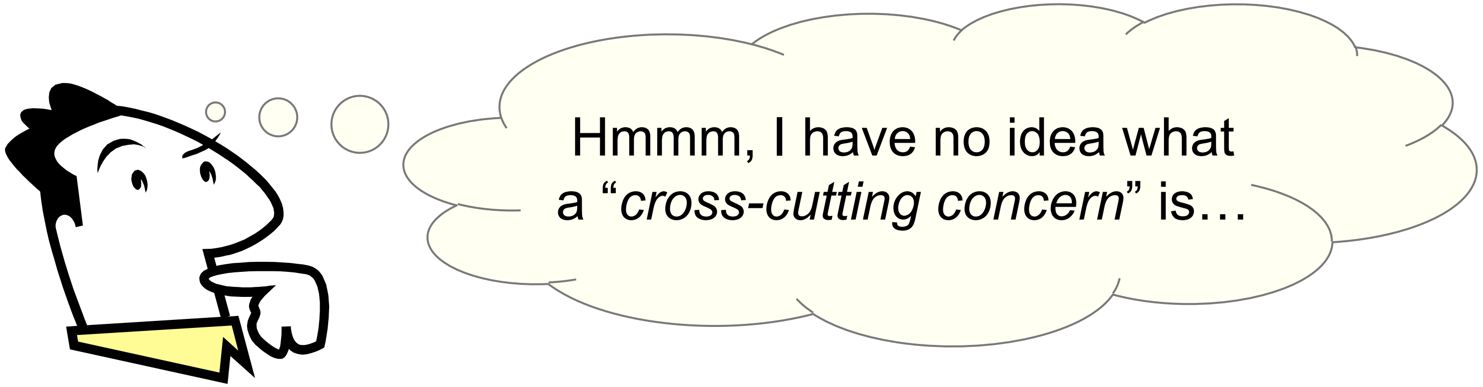
In addition to being an IoC object factory, the Spring Framework also provides an environment for ***Aspect Oriented Programming***

This is a programming paradigm that allows cross-cutting concerns to be separated into modular units known as ***Aspects***.

Aspect Oriented Programming

In addition to being an IoC object factory, the Spring Framework also provides an environment for ***Aspect Oriented Programming***

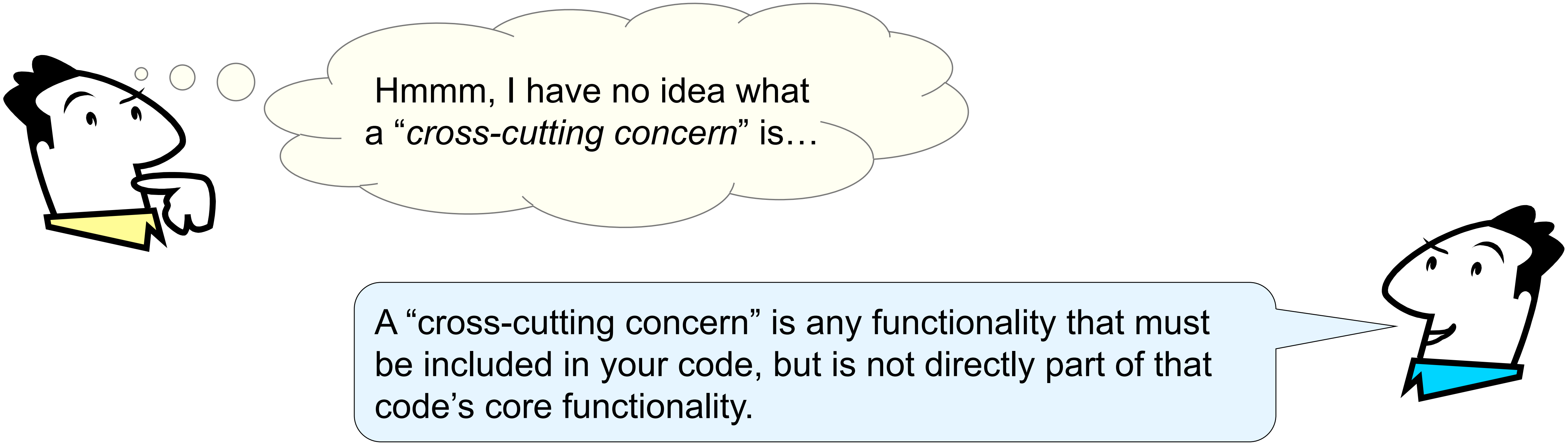
This is a programming paradigm that allows cross-cutting concerns to be separated into modular units known as ***Aspects***.



Aspect Oriented Programming

In addition to being an IoC object factory, the Spring Framework also provides an environment for ***Aspect Oriented Programming***

This is a programming paradigm that allows cross-cutting concerns to be separated into modular units known as ***Aspects***.


Hmmm, I have no idea what a "*cross-cutting concern*" is…
A "cross-cutting concern" is any functionality that must be included in your code, but is not directly part of that code's core functionality.

Cross-Cutting Concerns: Examples

A ***cross-cutting concern*** is any type of functionality that needs to be included in your code, but is not directly related to the code’s core functionality. In other words, these are units of functionality that refuse to conform to the abstractions used to implement a business task.

Cross-Cutting Concerns: Examples

A ***cross-cutting concern*** is any type of functionality that needs to be included in your code, but is not directly related to the code’s core functionality. In other words, these are units of functionality that refuse to conform to the abstractions used to implement a business task.

For example, if you are writing a Java object to handle Purchase Orders, you will need to write methods such as `create()` or `update()` . However, within these methods you will also need to include code to perform such additional tasks as:

Cross-Cutting Concerns: Examples

A ***cross-cutting concern*** is any type of functionality that needs to be included in your code, but is not directly related to the code's core functionality. In other words, these are units of functionality that refuse to conform to the abstractions used to implement a business task.

For example, if you are writing a Java object to handle Purchase Orders, you will need to write methods such as `create()` or `update()` . However, within these methods you will also need to include code to perform such additional tasks as:

- User authorization checks

Cross-Cutting Concerns: Examples

A ***cross-cutting concern*** is any type of functionality that needs to be included in your code, but is not directly related to the code's core functionality. In other words, these are units of functionality that refuse to conform to the abstractions used to implement a business task.

For example, if you are writing a Java object to handle Purchase Orders, you will need to write methods such as `create()` or `update()` . However, within these methods you will also need to include code to perform such additional tasks as:

- User authorization checks
- Logging

Cross-Cutting Concerns: Examples

A ***cross-cutting concern*** is any type of functionality that needs to be included in your code, but is not directly related to the code's core functionality. In other words, these are units of functionality that refuse to conform to the abstractions used to implement a business task.

For example, if you are writing a Java object to handle Purchase Orders, you will need to write methods such as `create()` or `update()` . However, within these methods you will also need to include code to perform such additional tasks as:

- User authorization checks
- Logging
- Internationalization

Cross-Cutting Concerns: Examples

A ***cross-cutting concern*** is any type of functionality that needs to be included in your code, but is not directly related to the code's core functionality. In other words, these are units of functionality that refuse to conform to the abstractions used to implement a business task.

For example, if you are writing a Java object to handle Purchase Orders, you will need to write methods such as `create()` or `update()` . However, within these methods you will also need to include code to perform such additional tasks as:

- User authorization checks
- Logging
- Internationalization
- Memory Management

Cross-Cutting Concerns: Examples

A ***cross-cutting concern*** is any type of functionality that needs to be included in your code, but is not directly related to the code's core functionality. In other words, these are units of functionality that refuse to conform to the abstractions used to implement a business task.

For example, if you are writing a Java object to handle Purchase Orders, you will need to write methods such as `create()` or `update()` . However, within these methods you will also need to include code to perform such additional tasks as:

- User authorization checks
- Logging
- Internationalization
- Memory Management
- Database Persistence

Cross-Cutting Concerns: Examples

A ***cross-cutting concern*** is any type of functionality that needs to be included in your code, but is not directly related to the code's core functionality. In other words, these are units of functionality that refuse to conform to the abstractions used to implement a business task.

For example, if you are writing a Java object to handle Purchase Orders, you will need to write methods such as `create()` or `update()` . However, within these methods you will also need to include code to perform such additional tasks as:

- User authorization checks
- Logging
- Internationalization
- Memory Management
- Database Persistence

None of these tasks are directly related to the activity of creating a purchase order insomuch as a purchase order ***could be*** created if all of these tasks were omitted.

Cross-Cutting Concerns: Examples

A ***cross-cutting concern*** is any type of functionality that needs to be included in your code, but is not directly related to the code's core functionality. In other words, these are units of functionality that refuse to conform to the abstractions used to implement a business task.

For example, if you are writing a Java object to handle Purchase Orders, you will need to write methods such as `create()` or `update()` . However, within these methods you will also need to include code to perform such additional tasks as:

- User authorization checks
- Logging
- Internationalization
- Memory Management
- Database Persistence

None of these tasks are directly related to the activity of creating a purchase order insomuch as a purchase order ***could be*** created if all of these tasks were omitted.

However, from the larger perspective of managing a business application system, these tasks are all vitally necessary.

Cross-Cutting Concerns: Existing Problems

Such coding performs tasks that are said to “cut across” the actual business functionality. Very often, the coding to implement cross-cutting concerns leads to the following architectural problems:

Cross-Cutting Concerns: Existing Problems

Such coding performs tasks that are said to “cut across” the actual business functionality. Very often, the coding to implement cross-cutting concerns leads to the following architectural problems:

- **Scattering**
  Cross-cutting code becomes widely scattered (and therefore duplicated) throughout your application. Any time scattered code needs to be changed, a large amount of effort is required which in turn, increases the chance of mistakes being made.

Cross-Cutting Concerns: Existing Problems

Such coding performs tasks that are said to “cut across” the actual business functionality. Very often, the coding to implement cross-cutting concerns leads to the following architectural problems:

- **Scattering**
  Cross-cutting code becomes widely scattered (and therefore duplicated) throughout your application. Any time scattered code needs to be changed, a large amount of effort is required which in turn, increases the chance of mistakes being made.
- **Tangling**
  The functionality of cross-cutting code becomes tangled with the functionality of the business coding. This can lead to unexpected behaviour if cross-cutting code is later modified.

Cross-Cutting Concerns: Existing Problems

Such coding performs tasks that are said to “cut across” the actual business functionality. Very often, the coding to implement cross-cutting concerns leads to the following architectural problems:

- **Scattering**
  Cross-cutting code becomes widely scattered (and therefore duplicated) throughout your application. Any time scattered code needs to be changed, a large amount of effort is required which in turn, increases the chance of mistakes being made.
- **Tangling**
  The functionality of cross-cutting code becomes tangled with the functionality of the business coding. This can lead to unexpected behaviour if cross-cutting code is later modified.

In addition to this, cross-cutting code does not fit cleanly into either the object-oriented or procedural programming paradigms.

Cross-Cutting Concerns: Existing Problems

Such coding performs tasks that are said to “cut across” the actual business functionality. Very often, the coding to implement cross-cutting concerns leads to the following architectural problems:

- **Scattering**
  Cross-cutting code becomes widely scattered (and therefore duplicated) throughout your application. Any time scattered code needs to be changed, a large amount of effort is required which in turn, increases the chance of mistakes being made.
- **Tangling**
  The functionality of cross-cutting code becomes tangled with the functionality of the business coding. This can lead to unexpected behaviour if cross-cutting code is later modified.

In addition to this, cross-cutting code does not fit cleanly into either the object-oriented or procedural programming paradigms.

Consequently, it is necessary to create a new programming paradigm in which cross-cutting code is separated into units known as ***Aspects***: hence the term Aspect Oriented Programming (AOP).

Cross-Cutting Concerns: Creating Aspects

In order to disentangle cross-cutting functionality from core business functionality, the AOP design process requires that you focus on two distinct properties of the cross-cutting functionality:

Cross-Cutting Concerns: Creating Aspects

In order to disentangle cross-cutting functionality from core business functionality, the AOP design process requires that you focus on two distinct properties of the cross-cutting functionality:

1. **WHAT**: Identify and isolate the functionality that cuts across the core business functionality

Cross-Cutting Concerns: Creating Aspects

In order to disentangle cross-cutting functionality from core business functionality, the AOP design process requires that you focus on two distinct properties of the cross-cutting functionality:

1. **WHAT**: Identify and isolate the functionality that cuts across the core business functionality
2. **WHEN**: Define the points in time at which those units of cross-cutting code should be invoked

Cross-Cutting Concerns: Creating Aspects

In order to disentangle cross-cutting functionality from core business functionality, the AOP design process requires that you focus on two distinct properties of the cross-cutting functionality:

1. **WHAT**: Identify and isolate the functionality that cuts across the core business functionality
2. **WHEN**: Define the points in time at which those units of cross-cutting code should be invoked

AOP uses the following terminology:

- ***Advice***
  Cross-cutting functionality is isolated into a unit called an "advice". The advice contains the "***what***".

Cross-Cutting Concerns: Creating Aspects

In order to disentangle cross-cutting functionality from core business functionality, the AOP design process requires that you focus on two distinct properties of the cross-cutting functionality:

1. **WHAT**: Identify and isolate the functionality that cuts across the core business functionality
2. **WHEN**: Define the points in time at which those units of cross-cutting code should be invoked

AOP uses the following terminology:

- ***Advice***
  Cross-cutting functionality is isolated into a unit called an "advice". The advice contains the "***what***".
- ***Join Point***
  The point in time at which an advice should be executed. A join point defines the "***when***".

Cross-Cutting Concerns: Creating Aspects

In order to disentangle cross-cutting functionality from core business functionality, the AOP design process requires that you focus on two distinct properties of the cross-cutting functionality:

1. **WHAT**: Identify and isolate the functionality that cuts across the core business functionality
2. **WHEN**: Define the points in time at which those units of cross-cutting code should be invoked

AOP uses the following terminology:

- ***Advice***
  Cross-cutting functionality is isolated into a unit called an "advice". The advice contains the "***what***".
- ***Join Point***
  The point in time at which an advice should be executed. A join point defines the "***when***".
- ***Pointcut***
  A "Pointcut" is the set of all join points related to a particular advice.

Cross-Cutting Concerns: Creating Aspects

In order to disentangle cross-cutting functionality from core business functionality, the AOP design process requires that you focus on two distinct properties of the cross-cutting functionality:

1. **WHAT**: Identify and isolate the functionality that cuts across the core business functionality
2. **WHEN**: Define the points in time at which those units of cross-cutting code should be invoked

AOP uses the following terminology:

- ***Advice***
  Cross-cutting functionality is isolated into a unit called an "advice". The advice contains the "***what***".
- ***Join Point***
  The point in time at which an advice should be executed. A join point defines the "***when***".
- ***Pointcut***
  A "Pointcut" is the set of all join points related to a particular advice.
- ***Aspect***
  An "Aspect" is the union of an advice and a pointcut.

Spring Dynamic Modules

Managing OSGi bundles using the Spring Framework

Spring Dynamic Modules: Spring Framework + OSGi

Up until now, we have talked only about the Spring Framework as it is used to manage individual Java objects (known as Beans). However, Spring places no restrictions on what type of objects it can manage, or what type of programming model should be used.

Spring Dynamic Modules: Spring Framework + OSGi

Up until now, we have talked only about the Spring Framework as it is used to manage individual Java objects (known as Beans). However, Spring places no restrictions on what type of objects it can manage, or what type of programming model should be used.

Therefore, if we use the Spring Framework to manage OSGi modules (a.k.a. bundles), then we arrive at a powerful set of tools for developing and managing OSGi bundles as if they were Java Beans.

Spring Dynamic Modules: Spring Framework + OSGi

Up until now, we have talked only about the Spring Framework as it is used to manage individual Java objects (known as Beans). However, Spring places no restrictions on what type of objects it can manage, or what type of programming model should be used.

Therefore, if we use the Spring Framework to manage OSGi modules (a.k.a. bundles), then we arrive at a powerful set of tools for developing and managing OSGi bundles as if they were Java Beans.

This is known as ***Spring Dynamic Modules*** *(or Spring DM)*.

See http://www.springsource.org/osgi for more details.

Spring Dynamic Module Overview 1/2

The combination of OSGi and the Spring Framework provides the following:

- Spring Framework JAR files as OSGi bundles
- Three additional Spring JAR files specific to OSGi
  - `org.springframework.osgi.bundle.extender`
  - `org.springframework.osgi.bundle.core`
  - `org.springframework.osgi.bundle.io`

Spring Dynamic Module Overview 2/2

When a Spring DM application starts, the following sequence of steps is performed:

Spring Dynamic Module Overview 2/2

When a Spring DM application starts, the following sequence of steps is performed:

1. The `org.springframework.osgi.bundle.extender` queries all existing bundles in the `resolved` state to see which ones are “Spring-powered”. This means identifying which bundles contain either:
   - A line starting with `Spring-context` in `META-INF/MANIFEST.MF`
   - Any XML files in the JAR location `META-INF/spring`

Spring Dynamic Module Overview 2/2

When a Spring DM application starts, the following sequence of steps is performed:

1. The `org.springframework.osgi.bundle.extender` queries all existing bundles in the `resolved` state to see which ones are “Spring-powered”. This means identifying which bundles contain either:
   - A line starting with `Spring-context` in `META-INF/MANIFEST.MF`
   - Any XML files in the JAR location `META-INF/spring`
2. For all Spring-powered bundles, the Spring configuration is loaded and normal Spring object management processing takes place

Spring Dynamic Module Overview 2/2

When a Spring DM application starts, the following sequence of steps is performed:

1. The `org.springframework.osgi.bundle.extender` queries all existing bundles in the `resolved` state to see which ones are "Spring-powered". This means identifying which bundles contain either:
   - A line starting with `Spring-context` in `META-INF/MANIFEST.MF`
   - Any XML files in the JAR location `META-INF/spring`
2. For all Spring-powered bundles, the Spring configuration is loaded and normal Spring object management processing takes place
3. The `extender` checks to see if the bundle imports or exports any OSGi services. If so, exported services are published as Spring beans to the OSGi service registry and imported service dependencies are resolved.

Spring Dynamic Module Overview 2/2

When a Spring DM application starts, the following sequence of steps is performed:

1. The `org.springframework.osgi.bundle.extender` queries all existing bundles in the `resolved` state to see which ones are "Spring-powered". This means identifying which bundles contain either:
    - A line starting with `Spring-context` in `META-INF/MANIFEST.MF`
    - Any XML files in the JAR location `META-INF/spring`
2. For all Spring-powered bundles, the Spring configuration is loaded and normal Spring object management processing takes place
3. The `extender` checks to see if the bundle imports or exports any OSGi services. If so, exported services are published as Spring beans to the OSGi service registry and imported service dependencies are resolved.
4. The `extender` also registers a bundle listener to react should the bundle ever change back into the `resolved` state.

Summary

Spring Framework: Summary

The Spring Framework is an object factory that provides a runtime environment for the management of Java objects using Inversion of Control (implemented through Dependency Injection).

However, the Spring Framework is modular and provides additional functionality for:

- Aspect Oriented Programming
- Data Access (JDBC, Hibernate, JDO, Apache Cayenne etc.)
- Transaction Management
- Model-View-Controller
- Remote Access
- Authentication
- Messaging
- Remote Management
- Testing

Aspect Oriented Programming: Summary 1/2

Aspect Oriented Programming (AOP) is programming paradigm that separates cross-cutting concerns from core business functionality.

A cross-cutting concern is any type functionality needed for the successful running of an enterprise system, but in itself, is not the core business functionality performed by that system. For example:

- User authorization checks
- Logging
- Internationalization
- Memory Management
- Database Persistence

Aspect Oriented Programming: Summary 2/2

AOP focuses on identifying the "what" and the "when" of cross-cutting functionality. It then handles these units of code in a manner that integrates them with the core business functionality whilst at the same time, avoids them becoming tangled in it.

AOP Terminology

Advice: ***What*** needs to be done

Join Point: ***When*** it needs to be done

Pointcut: The set of join points related to a particular advice

Aspect: The combination of an Advice and a Pointcut

Spring Dynamic Modules: Summary

Spring Dynamic Modules (Spring DM) combines the object factory capabilities of the Spring Framework with the large scale bundles of business functionality defined in OSGi.

This allows us to develop and manage OSGi bundles as if they were Java Beans.

For a free eBook on Spring DM, visit http://it-ebooks.info/book/1907/

© 2013 SAP AG. All rights reserved

This document is a preliminary version and not subject to your license agreement or any other agreement with SAP. This document contains only intended strategies, developments, and functionalities of the SAP® product and is not intended to be binding upon SAP to any particular course of business, product strategy, and/or development. Please note that this document is subject to change and may be changed by SAP at any time without notice.

SAP assumes no responsibility for errors or omissions in this document. SAP does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

SAP shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials. This limitation shall not apply in cases of intent or gross negligence.

The statutory liability for personal injury and defective products is not affected. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third-party Web pages nor provide any warranty whatsoever relating to third-party Web pages.

© 2013 SAP AG. Alle Rechte vorbehalten.

Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die ausdrückliche schriftliche Genehmigung durch SAP AG nicht gestattet. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Die von SAP AG oder deren Vertriebsfirmen angebotenen Softwareprodukte können Softwarekomponenten auch anderer Softwarehersteller enthalten.

Microsoft, Windows, Excel, Outlook, und PowerPoint sind eingetragene Marken der Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, z10, z/VM, z/OS, OS/390, zEnterprise, PowerVM, Power Architecture, Power Systems, POWER7, POWER6+, POWER6, POWER, PowerHA, pureScale, PowerPC, BladeCenter, System Storage, Storwize, XIV, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, AIX, Intelligent Miner, WebSphere, Tivoli, Informix und Smarter Planet sind Marken oder eingetragene Marken der IBM Corporation.

Linux ist eine eingetragene Marke von Linus Torvalds in den USA und anderen Ländern.

Adobe, das Adobe-Logo, Acrobat, PostScript und Reader sind Marken oder eingetragene Marken von Adobe Systems Incorporated in den USA und/oder anderen Ländern.

Oracle und Java sind eingetragene Marken von Oracle und/oder ihrer Tochtergesellschaften.

UNIX, X/Open, OSF/1 und Motif sind eingetragene Marken der Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame und MultiWin sind Marken oder eingetragene Marken von Citrix Systems, Inc.

HTML, XML, XHTML und W3C sind Marken oder eingetragene Marken des W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Apple, App Store, iBooks, iPad, iPhone, iPhoto, iPod, iTunes, Multi-Touch, Objective-C, Retina, Safari, Siri und Xcode sind Marken oder eingetragene Marken der Apple Inc.

IOS ist eine eingetragene Marke von Cisco Systems Inc.

RIM, BlackBerry, BBM, BlackBerry Curve, BlackBerry Bold, BlackBerry Pearl, BlackBerry Torch, BlackBerry Storm, BlackBerry Storm2, BlackBerry PlayBook und BlackBerry App World sind Marken oder eingetragene Marken von Research in Motion Limited.

Google App Engine, Google Apps, Google Checkout, Google Data API, Google Maps, Google Mobile Ads, Google Mobile Updater, Google Mobile, Google Store, Google Sync, Google Updater, Google Voice, Google Mail, Gmail, YouTube, Dalvik und Android sind Marken oder eingetragene Marken von Google Inc.

INTERMEC ist eine eingetragene Marke der Intermec Technologies Corporation.

Wi-Fi ist eine eingetragene Marke der Wi-Fi Alliance.

Bluetooth ist eine eingetragene Marke von Bluetooth SIG Inc.

Motorola ist eine eingetragene Marke von Motorola Trademark Holdings, LLC.

Computop ist eine eingetragene Marke der Computop Wirtschaftsinformatik GmbH.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP BusinessObjects Explorer, StreamWork, SAP HANA und weitere im Text erwähnte SAP-Produkte und -Dienstleistungen sowie die entsprechenden Logos sind Marken oder eingetragene Marken der SAP AG in Deutschland und anderen Ländern.

Business Objects und das Business-Objects-Logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius und andere im Text erwähnte Business-Objects-Produkte und -Dienstleistungen sowie die entsprechenden Logos sind Marken oder eingetragene Marken der Business Objects Software Ltd. Business Objects ist ein Unternehmen der SAP AG.

Sybase und Adaptive Server, iAnywhere, Sybase 365, SQL Anywhere und weitere im Text erwähnte Sybase-Produkte und -Dienstleistungen sowie die entsprechenden Logos sind Marken oder eingetragene Marken der Sybase Inc. Sybase ist ein Unternehmen der SAP AG.

Crossgate, m@gic EDDY, B2B 360°, B2B 360° Services sind eingetragene Marken der Crossgate AG in Deutschland und anderen Ländern. Crossgate ist ein Unternehmen der SAP AG.

Alle anderen Namen von Produkten und Dienstleistungen sind Marken der jeweiligen Firmen. Die Angaben im Text sind unverbindlich und dienen lediglich zu Informationszwecken. Produkte können länderspezifische Unterschiede aufweisen.

Die in dieser Publikation enthaltene Information ist Eigentum der SAP. Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, nur mit ausdrücklicher schriftlicher Genehmigung durch SAP AG gestattet.