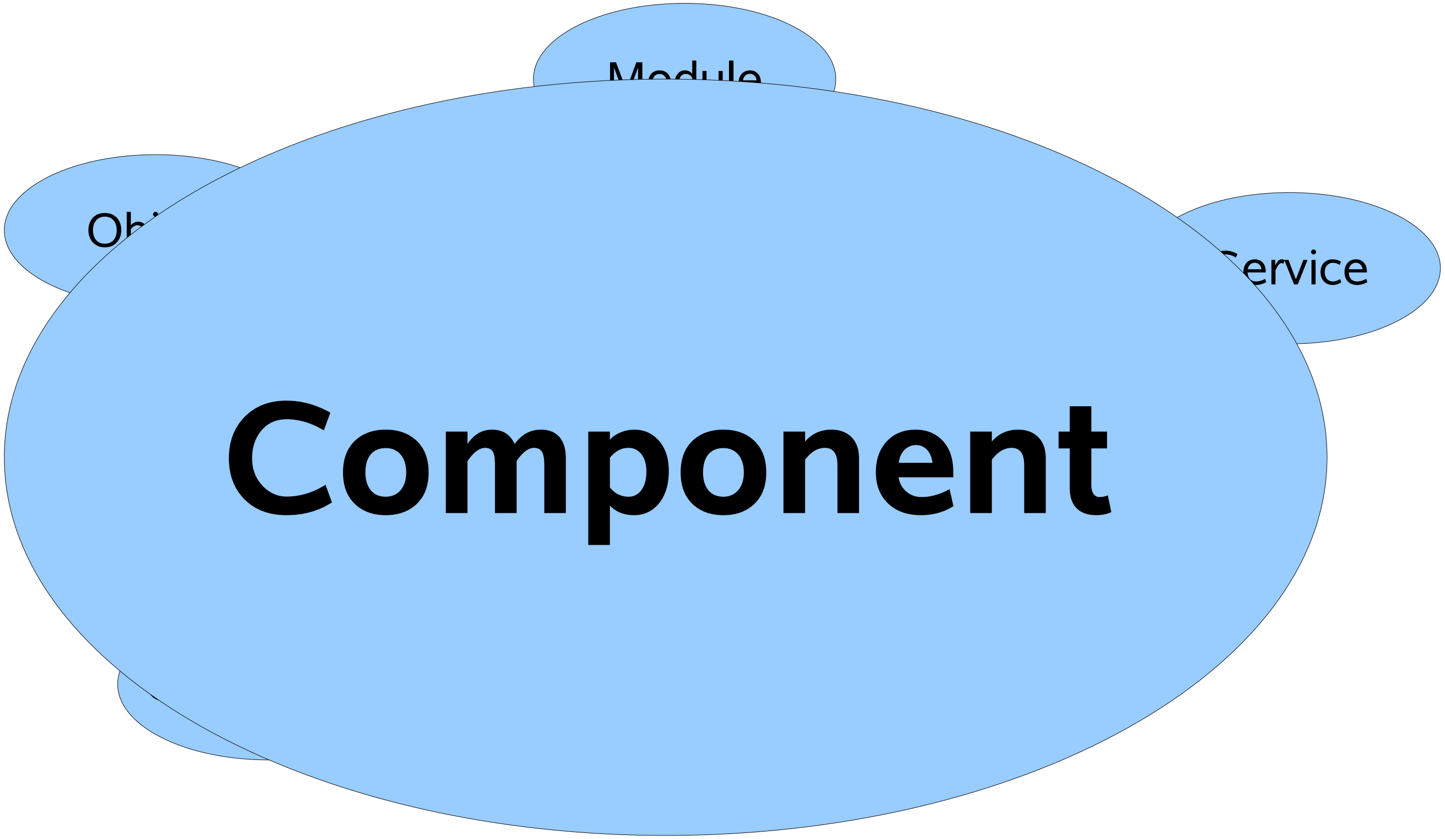
Stop writing `new`

A Comparison of Dependency Injection Frameworks

Erik Hellman <erik.hellman@ibs.se>
IBS JavaSolutions AB

IBS JavaSolutions



Module

Obj

Service

Component

Component Assembly 1: “Old style”

```
public class MyFirstComponent {
  private MySecondComponent component;


  public MyFirstComponent() {
    component = new
     MySecondComponent(param1, param2,
     param3, ...);
  }
}
```

Component Assembly 2: Factory

```
public class MyFirstComponent {
  private MySecondComponent component;

  public MyFirstComponent() {
    component =
     MyFactory.newMySecondComponent();
  }
}
```

Component Assembly 3: Service Locator

```
public class MyFirstComponent {
  private MySecondComponent component;

  public MyFirstComponent() {
     component = (MySecondComponent)
      InitialContext.
        lookup(“comp/MySecondComponent”);
  }
}
```

Component Assembly 4: Dependency Injection (setter)

```
public class MyFirstComponent {

    private MySecondComponent component;


    public MyFirstComponent() { }


    public void setComponent(MySecondComponent
      component) {
      this.component = component;

    }

}
```

Component Assembly 5: Dependency Injection (annotation)

```
public class MyFirstComponent {
   @Inject(Scope.SINGLETON)
   private MySecondComponent component;

   public MyFirstComponent() { }

}
```

Why Dependency Injection?

- *Separation of concerns*
- Encourage component-oriented design
- Easier unit-testing
- Simplify maintenance
- Fewer lines of code

Dependency Injection with Annotations

- Existing frameworks:
  - Spring Framework
  - Java EE 5/EJB 3
  - JBoss Seam
  - Google Guice

Java EE 5 and EJB 3

- Official Java standard (i.e., multiple JSR:s)
- Supported by most major Java Application Servers
- Excellent support in the three major IDE:s
- Requires a Java EE container
- Unit-testing is a little complicated
- Dependey Injection limited to EJBs and JavaEE components
- “Boilerplate code” required for presentation layer (JSF, Struts2 etc.)

Spring Framework

- Lots and lots of utilities
- No Java EE requirement
- Plug-ins for all major IDE:s
- Excellent (best?) documentation
- Simple unit-testing
- Easy to integrate with other frameworks
- Many dependencies on additional 3PP
- Steep learning-curve
- Easy to make severe mistakes

JBoss Seam

- “Deep Integration Framework”
- Bridges the gap between JSF and EJB 3
- Focused on web applications
- Extremely easy to use
- Requires no additional 3PP or frameworks (in most cases)
- Performance issues!
- Only supporting JSF for presentation technologies
- Hard to read documentation

Google Guice

- “Ultra lightweight” - extremely dynamic
- No Java EE requirements
- Struts 2 plug-in
- Java 5 or later
- Good documentation, tutorials and examples
- Easy to get started
- Google AdWords!

“Homebrew” (1)

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface Inject {
}

public class ComponentOne {

    @Inject
    private ComponentTwo componentTwo;

    @Inject
    private ComponentThree componentThree;
    private String name;
    private long id;

    public static void main(String[] args) {
        ComponentOne componentOne = new ComponentOne();
        componentOne.setName("firstComponent");
        componentOne.setId(1l);
        Injector.getInstance().doInjection(componentOne);

        System.out.println("Done injecting!");
    }

}
```

“Homebrew” (2)

```
public void doInjection(Object target) {
    Field[] fields = target.getClass().getDeclaredFields();
    for (Field field : fields) {
        Annotation[] annotations = field.getDeclaredAnnotations();
        Class componentClass = field.getType();
        for (Annotation annotation : annotations) {
            if(annotation.annotationType().equals(Inject.class)) {
                Object componentObject = singletons.get(componentClass);
                if(componentObject == null) {
                    try {
                        componentObject = componentClass.newInstance();
                        singletons.put(componentClass, componentObject);
                        doInjection(componentObject);
                    } catch (InstantiationException e) {
                        e.printStackTrace();
                    } catch (IllegalAccessException e) {
                        e.printStackTrace();
                    }
                }
                try {
                    field.setAccessible(true);
                    field.set(target, componentObject);
                } catch (IllegalAccessException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Conclusions

- Type of application?
- Limitations (organizational, technical, knowledge)?
- Integration with external systems?
- Commercial product?

Thanks for listening!

IBS JavaSolutions