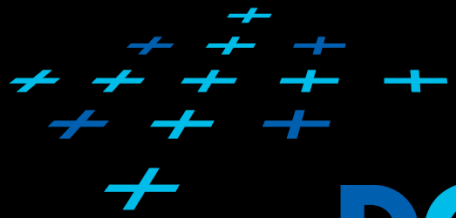


DCGI

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

# Annotations in Java

Martin Klíma

# Annotations

- Additional processing instructions
- Written by programmer in the source code
- Used by an external entity – container, context

```
@Resource(name = "customerDB")
   public void setDataSource(DataSource myDB) {
      this.ds = myDB;
   }
@EJB
   public ShoppingCart myShoppingCart;

 @Local
   public interface RepeaterSessionBeanLocal {
   }

 @Copyright("2002 Yoyodyne Propulsion Systems")
   public class OscillationOverthruster {
      ...
   }
```

# Annotations are defined by the construct

```
public @interface RequestForEnhancement {
    int    id();
    String synopsis();
    String engineer() default "[unassigned]";
    String date()  default "[unimplemented]";
}
```

```
public class EnhancementTest {

    @RequestForEnhancement(id = 2868724,
    synopsis = "Enable time-travel",
    engineer = "Mr. Peabody",
    date = "4/1/3007")
    public static void travelThroughTime(Date destination) {
        // do something here
    }
}
```

# A complex example

taken from http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

Annotations = metadata

## Annotated code

```
public class Foo {
   @Test public static void m1() { }
   public static void m2() { }
   @Test public static void m3() {
      throw new RuntimeException("Boom");
   }
   public static void m4() { }
   @Test public static void m5() { }
   public static void m6() { }
   @Test public static void m7() {
      throw new RuntimeException("Crash");
   }
   public static void m8() { }
}
```

## Use of annotation by a container

```
public class RunTests {
  public static void main(String[] args) throws Exception
{
    int passed = 0, failed = 0;
    for (Method m :
Class.forName(args[0]).getMethods()) {
      if (m.isAnnotationPresent(Test.class)) {
        try {
          m.invoke(null);   passed++;
        } catch (Throwable ex) {
          System.out.printf("Test %s failed: %s %n", m,
ex.getCause());   failed++;
        }
      }
    }
    System.out.printf("Passed: %d, Failed %d%n",
passed, failed);
  }
}
```

Use of reflection

# More details about annotations

- Default values can be defined
- Some annotation do already exist in Java
  - @Retention
    - SOURCE (source code only), CLASS (in binary class), RUNTIME (in runtime)
  - @Target – enumeration ElementType
    - TYPE
    - FIELD
    - METHOD
    - PARAMETER
    - CONSTRUCTOR
    - LOCAL_VARIABLE
    - ANNOTATION_TYPE
    - PACKAGE
  - @Inherited
    - Descendants of annotated class are also annotated



# Annotations last one

- Annotations without a value

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

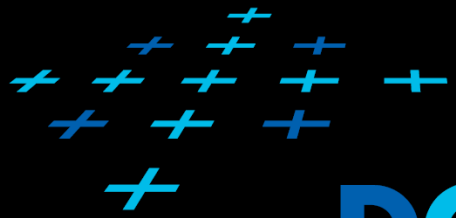
- Annotation with one value

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Inherited
/**
 * Trida bude vracet chybovy stav uvedeny ve {@code value}.
 */
public @interface ErrorPage {
    int value();
}
```

- With multiple values and a default one

```
public @interface RequestForEnhancement {
    int    id();
    String synopsis();
    String engineer() default "[unassigned]";
    String date()  default "[unimplemented]";
}
```





DCGI

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

# Inversion of Control

WA 2

Martin Klíma

# What is Inversion of Control

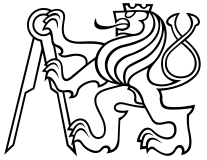
- IoC is a design paradigm
- Decoupling the execution of a task from implementation.
- Focus on what you (module) have to do.
- No assumption about other systems. Rely on contracts only.
- Tight coupling – replacing other modules has not effect.



# Dependency Injection

- One of the ways to implement IoC.
- Create instances of objects when needed, NOT at compile time.





# How IoC is implemented

1. Factory pattern
2. Service locator
3. Dependency injection
   - Constructor injection
   - Setter injection
   - Interface injection



# Now some examples (http://martinfowler.com/articles/injection.html)

Lets have a simple and naive program that find all movies produced by a given director…

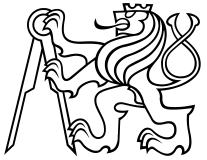
```
class MovieLister...
  public Movie[] moviesDirectedBy(String arg) {

      List allMovies = finder.findAll();

      for (Iterator it = allMovies.iterator(); it.hasNext();) {
          Movie movie = (Movie) it.next();
          if (!movie.getDirector().equals(arg)) it.remove();
      }
      return (Movie[]) allMovies.toArray(new Movie[allMovies.size()]);
  }
```

A finder should be independent on my code. Finder is not known at compilation time

DCGI

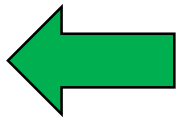


# OK, let's have a contract

```
public interface MovieFinder {
    List findAll();
}
```

## …and the MovieLister can now look like this:

```
class MovieLister...
  private MovieFinder finder;
  public MovieLister() {
    finder = new ColonDelimitedMovieFinder("movies1.txt");
  }
```



Tight coupling.
Any way out?

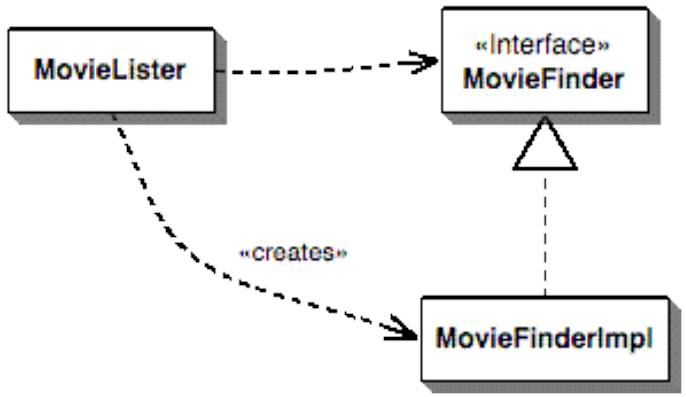

# Ways out of tight coupling

- Factory – looser but still too tight (compile time)
- Plug-in

```
private static Object buildObject(String aClassName){
    Object result = null;
   try {
      Class implClass = Class.forName(aClassName);
      result = implClass.newInstance();
   }
   catch (ClassNotFoundException ex) {

}
   catch (InstantiationException ex) {
}
   catch (IllegalAccessException ex) {

}
   return result;
  }
}
```

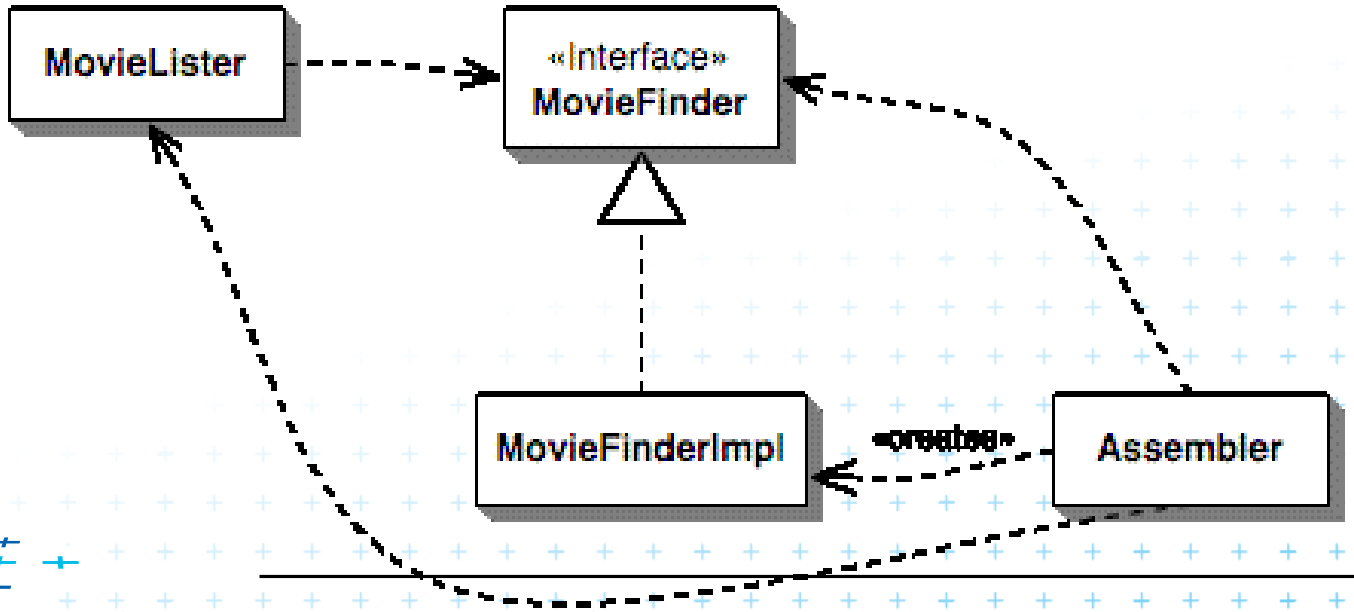



(no compile time)

- IoC (no compile time)

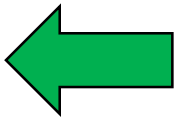
# Inversion of Control

- The MovieLister should look up the finder.
- Some external entity – an assembler – should provide it.
- The assembler can be configured to provide different implementation if needed.



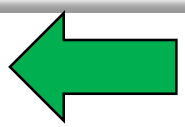
# Example of PicoContainer DI

```
public MovieLister(MovieFinder finder) {
    this.finder = finder;
...
}
```



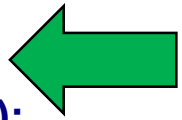
Loose coupling

```
class ColonMovieFinder...
  public ColonMovieFinder(String filename) {
    this.filename = filename;
...
}
```



One implementing class

```
private MutablePicoContainer configureContainer() {
    MutablePicoContainer pico = new DefaultPicoContainer();
    Parameter[] finderParams =  {new ConstantParameter("movies1.txt")};
    pico.registerComponentImplementation(MovieFinder.class, ColonMovieFinder.class,
finderParams);
    pico.registerComponentImplementation(MovieLister.class);
    return pico;
}
```

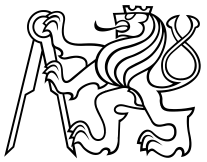


Container setup (can be provided as XML)

# Usage

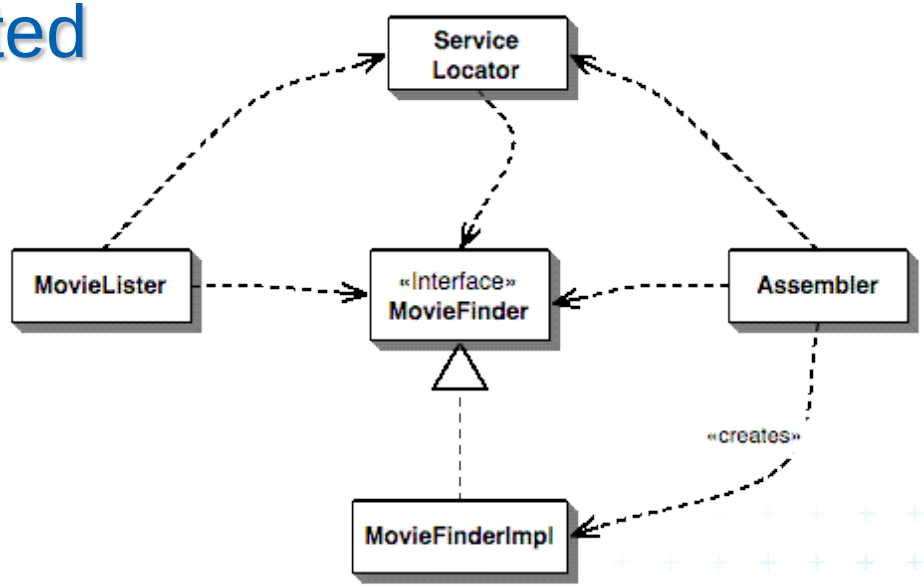
```
public void testWithPico() {
    MutablePicoContainer pico = configureContainer();
    MovieLister lister = (MovieLister) pico.getComponentInstance(MovieLister.class);
    Movie[] movies = lister.moviesDirectedBy("Sergio Leone");
    assertEquals("Once Upon a Time in the West", movies[0].getTitle());
}
```





# Service Locator using DI

- Lets have a service-oriented architecture
- Services register to **Registry**



# Static implementation of Service Locator

```
class MovieLister...

  MovieFinder finder = ServiceLocator.movieFinder();
```

```
class ServiceLocator...

private static ServiceLocator soleInstance;
private MovieFinder movieFinder;

public static MovieFinder movieFinder() {
      return soleInstance.movieFinder;
}

public static void load(ServiceLocator arg) {
      soleInstance = arg;
}

public ServiceLocator(MovieFinder movieFinder) {
      this.movieFinder = movieFinder;
  }
```

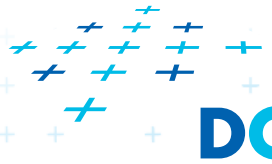


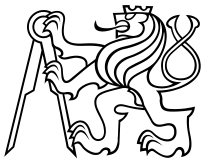
# Try it out

```
class Tester...

private void configure() {
    ServiceLocator.load(new ServiceLocator(new ColonMovieFinder("movies1.txt")));
}

public void testSimple() {
    configure();
    MovieLister lister = new MovieLister();
    Movie[] movies = lister.moviesDirectedBy("Sergio Leone");
    assertEquals("Once Upon a Time in the West", movies[0].getTitle());
}
```



# We usually want a dynamic ServiceLocator

```
class ServiceLocator...
  private static ServiceLocator soleInstance;
  private Map services = new HashMap();

  public static void load(ServiceLocator arg) {
      soleInstance = arg;
  }

public static Object getService(String key){
      return soleInstance.services.get(key);
  }

  public void loadService (String key, Object service) {
      services.put(key, service);
  }
```



# A dynamic ServiceLocator can be configured

```
class Tester...

  private void configure() {
      ServiceLocator locator = new ServiceLocator();
      locator.loadService("MovieFinder", new ColonMovieFinder("movies1.txt"));
      ServiceLocator.load(locator);
  }
```

Use in MovieLister

```
class MovieLister...

  MovieFinder finder = (MovieFinder) ServiceLocator.getService("MovieFinder");
```