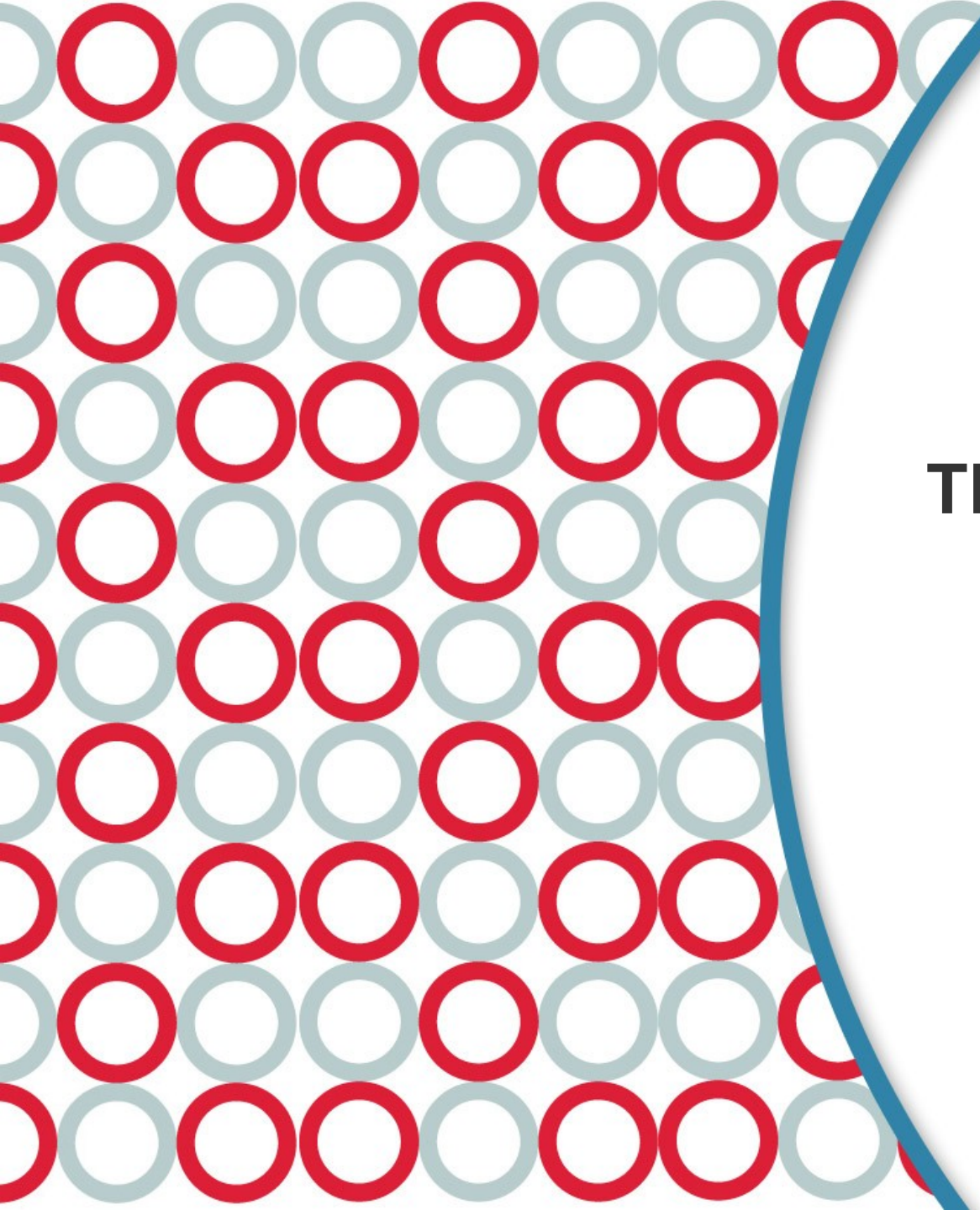


May 2019, IPT Course
Java Web Debelopment

The Reflection API

Trayan Iliev
tiliev@iproduct.org
http://iproduct.org

Where to Find the Code?

Intarmediate Java Programming projects and examples are available @ GitHub:

https://github.com/iproduct/course-java-web-development

RTTI & Reflection

- **Runtime Type Information (RTTI)** allows you to discover and employ type information in runtime.
- The difference between **RTTI** and **reflection** is that with **RTTI**, the compiler **opens and examines the .class file at compile time**, while with **reflection**, the .class file is unavailable at compile time; it is **opened and examined by the runtime** environment. Examples:
  - **JavaBeans** - Rapid Application Development (RAD) in an Application Builder Integrated Development Environment (IDE)
  - Object serialization – Serializable interface
  - **Remote Method Invocation (RMI)** - discovering class information at run time provides ability to create and execute objects on remote platforms, across the network.
  - **Dynamic Proxies** – DI, Spring, AOP

Runtime Type Information (RTTI)

- The **Class** object, **Class.forName()**, **Type.class**, and **object.getClass( )**
- **Class** models all types in java – class, interface, array, primitive type, void (e.g. int.class, long.class, double.class, void.class, etc.)
- It allows writing flexible and generic utility methods and tools capable of processing (field, method, type variable, annotation, superclass and interface metadata reflection, instantiation (constructor invocation), method invocation, field data access, etc.) of different types of objects given as parameters to those methods and tools, and generally not known at the time of their writing.
- **T newInstance()** - creates a new instance of the class
- **T cast(Object obj)** - casts an object to the class or interface

Class: Names, Loader, Annotations

- **String getName()** - name of the type (class, interface, array, primitive type, void)
- **String getSimpleName()** - the simple name of the underlying class
- **String getCanonicalName()** - canonical name of class
- **ClassLoader getClassLoader()** - returns the class loader for the class.
- **<A extends Annotation>A getAnnotation(Class<A> annotationClass)** - annotation if present
- **Annotation[] getAnnotations()** - all annotations present on this type
- **<A extends Annotation> A[] getAnnotationsByType(Class<A> annotationClass)** – if its argument is a repeatable annotation type (JLS 9.6), it attempts to find one or more annotations of that type by "looking through" a container annotation.
- **Annotation[] getDeclaredAnnotations()** - directly present annotations
- **<A extends Annotation> A getDeclaredAnnotation(Class<A> annotationClass)**
- **<A extends Annotation> A[] getDeclaredAnnotationsByType(Class<A>)**
- **boolean isAnnotation()** - true if an annotation type.
- **Boolean isAnnotationPresent(Class<? extends Annotation> annotatClass)**

Class: Constructors, Fields, Methods

- **Constructor<?>[] getConstructors()** - array of all public Constructors
- **Constructor<T> getConstructor(Class<?>... parameterTypes)** – public
- **Constructor<?>[] getDeclaredConstructors()** - all class constructors
- **Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes)**
- **Field getField(String name)** - public member field
- **Field[] getFields()** - public member fields
- **Field getDeclaredField(String name)** - a field:public, private, package, protected
- **Field[] getDeclaredFields()** - all class fields: public, private, package, protected
- **Method getMethod(String name, Class<?>... parameterTypes)** - reflects the specified public member method
- **Method[] getMethods()** - reflects all public member method
- **Method getDeclaredMethod(String name, Class<?>... parameterTypes)**
- **Method[] getDeclaredMethods()** - all methods

Class: Members, Inner Classes, Types

- **Class<?>[] getClasses()** - public class and interface members of this Class
- **Class<?>[] getDeclaredClasses()** - all the class and interface members
- **Class<?> getDeclaringClass()** - if a member of another class
- **Class<?> getEnclosingClass()** - enclosing class of the underlying class
- **Constructor<?> getEnclosingConstructor()** - for local/anonymous class
- **Method getEnclosingMethod()** - for local/anonymous class
- **T[] getEnumConstants()** - elements of this enum class
- **Type[] getGenericInterfaces()** - Types representing the interfaces impl.
- **Type getGenericSuperclass()** - Type representing the superclass impl.
- **AnnotatedType[] getAnnotatedInterfaces()** - annotated superinterfaces
- **AnnotatedType getAnnotatedSuperclass()** - annotated superclass
- **Class<?> getComponentType()** - the component type of an array

Class: SuperClasses, Interfaces, Resources

- **Class<? super T> getSuperclass()** - the superclass of the entity
- **Class<?>[] getInterfaces()** - the interfaces implemented
- **int getModifiers()** - Java language modifiers for entity (public, package) - as int
- **Package getPackage()** - gets the package for this class.
- **URL getResource(String name)** - finds a resource with a given name
- **InputStream getResourceAsStream(String name)** - finds a resource with a given name.
- **ProtectionDomain getProtectionDomain()** - returns the ProtectionDomain of this class.
- **Object[] getSigners()** - gets the signers of this class
- **String toGenericString()** - Returns a string describing this Class, including information about modifiers and type parameters

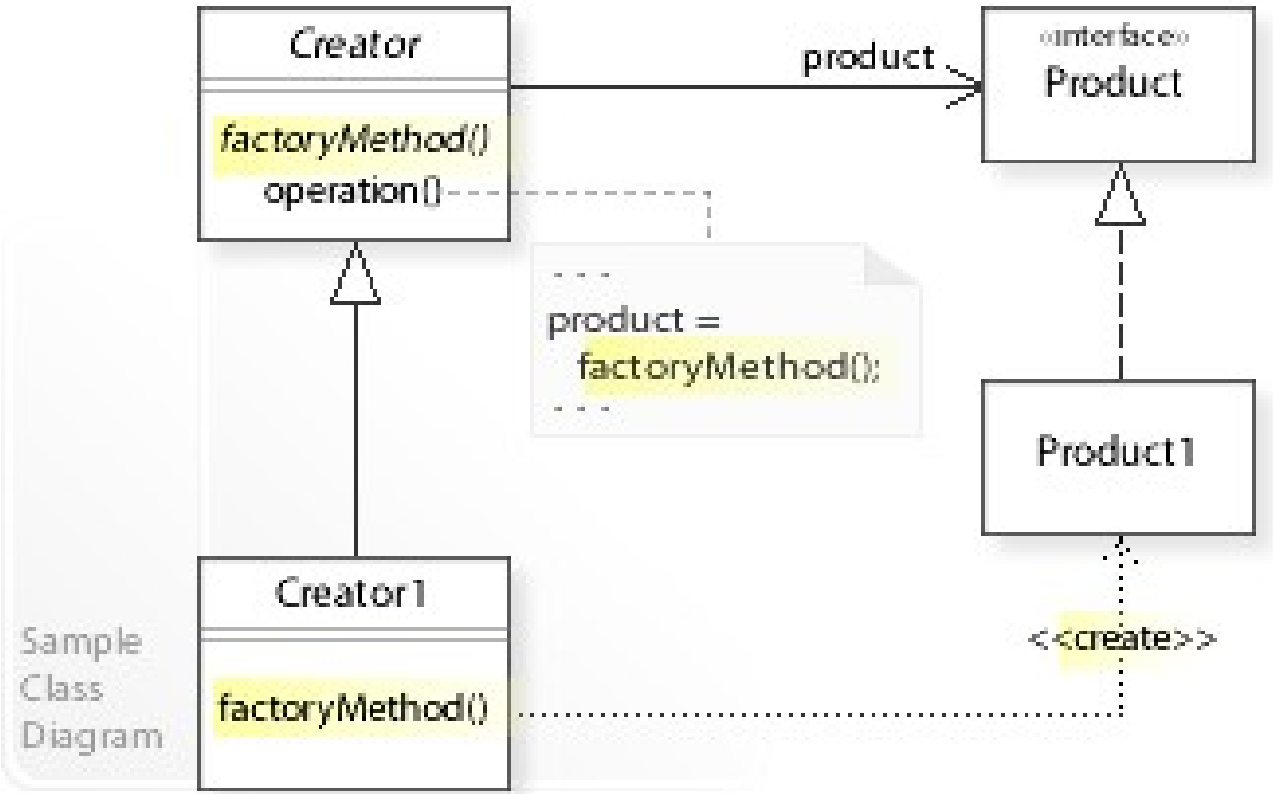
Class: Superclass, Type Vars, Is*

- **String getTypeName()** - an informative string for the name of this type.
- **TypeVariable<Class<T>>[] getTypeParameters()** - generic declaration GenericDeclaration TypeVariable objects
- **boolean isAnonymousClass()** - true if anonymous class.
- **boolean isArray()** - if an array class.
- **boolean isAssignableFrom(Class<?> cls)** -same/ superclass/ superinterface
- **boolean isEnum()** - if enum
- **boolean isInstance(Object obj)** – dynamic instanceof
- **boolean isInterface()** - if interface
- **boolean isLocalClass()** - if local class
- **boolean isMemberClass()** - if the underlying class is a member of this class
- **boolean isPrimitive()** - if the specified Class object represents a primitive type
- **boolean isSynthetic()** - if this class is a synthetic class

Factory Method Design Pattern

- **Factory method** pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created.
- This is done by creating objects by calling a factory method – either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes – rather than by calling a constructor.

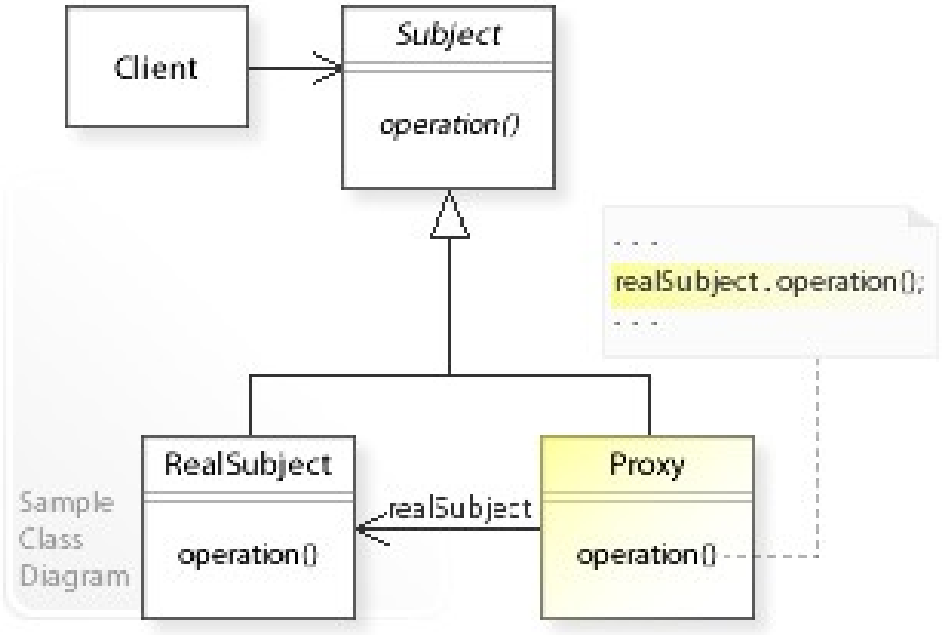
Factory Method Design Pattern

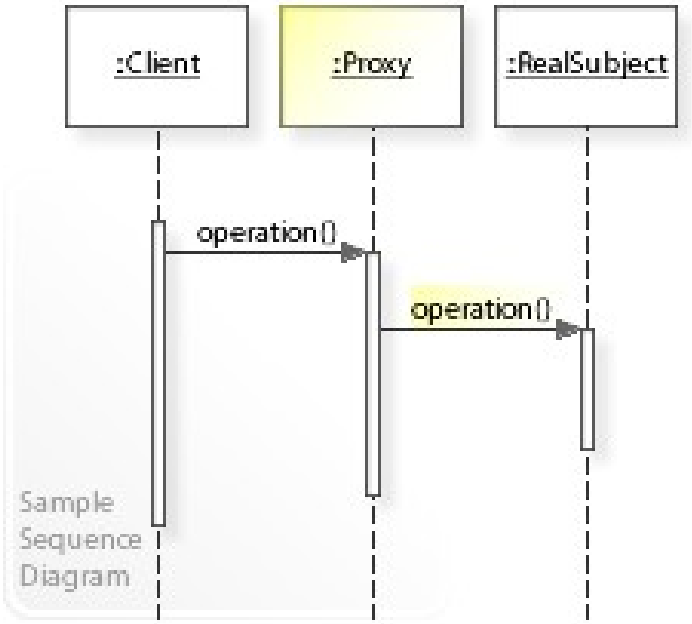


Dynamic Proxy Design Pattern

- A proxy, in its most general form, is a class functioning as an **interface to something else**. A proxy is a wrapper or agent object that is being called by the client to access the real serving object behind the scenes. Use of the proxy can simply be forwarding to the real object, or can provide **additional logic**.
- In the proxy, **extra functionality can be provided**, for example caching when operations on the real object are resource intensive, or checking preconditions before operations on the real object are invoked.
- For the client, usage of a proxy object is similar to using the real object, because **both implement the same interface**.
- **Dynamic proxies** can be generated through reflection of bean methods, providing additional functionality

Dynamic Proxy Design Pattern





Thank's for Your Attention!

Trayan Iliev

CEO of IPT – Intellectual Products

& Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD

https://plus.google.com/+IproductOrg