

sPearCat-Ready Superclass Template

```
#if ($entity.packageName)
package $entity.superclassPackageName;
#end

import com.weboobjects.foundation.*;
#set ($useControl = "false")
#if (!$entity.sortedClassToManyRelationships.empty)
#set ($useControl = "true")
#elseif (!$entity.sortedFetchSpecs.empty)
#set ($useControl = "true")
#elseif (!$entity.isAbstract)
#set ($useControl = "true")
#elseif (!$entity.parentSet)
#set ($useControl = "true")
#end
#if ($useControl == "true")
import com.weboobjects.eocontrol.*;
#end
#set ($useBigDecimal = "false")
#foreach ($attribute in $entity.sortedClassAttributes)
#if (!$attribute.inherited)
#if ($attribute.javaClassName == "BigDecimal") #set ($useBigDecimal = "true")#end
#end
#end
#if ($useBigDecimal == "true")
import java.math.BigDecimal;
#end
#if (!$entity.sortedClassToManyRelationships.empty)
import java.util.*;
#end

/**
 * Java Class ${entity.prefixClassNameWithOptionalPackage}.java generated for entity ${entity.name}
 *if ($entity.parentSet)
 * <p>
 * The entity parent is ${entity.parent.classNameWithDefault}
 * </p>
#end
 * <p><b>
 * DO NOT EDIT. Make changes to ${entity.classNameWithOptionalPackage}.java instead.
 * </b></p>
 * <p>
 * copyright ${copyrightYear} ${copyrightBy}
 * </p>
 */
public abstract class ${entity.prefixClassNameWithoutPackage} extends#if ($entity.parentSet) ${entity.parent.
classNameWithDefault}#else EOGenericRecord#end {
    private static org.apache.log4j.Logger logger = org.apache.log4j.Logger.getLogger( ${entity.
prefixClassNameWithoutPackage}.class);
#foreach ($attribute in $entity.sortedClassAttributes)
#if (!$attribute.inherited)

        /**
         * Key value for attribute $attribute.name
         */
        public static final String ${attribute.capitalizedName}Key = "$attribute.name";
#end
#end
// Inherited attribute $attribute.name
#end
#end
#foreach ($relationship in $entity.sortedClassToOneRelationships)
#if (!$relationship.inherited)

        /**
         * Key value for to one relationship $relationship.name
```

```

        */
        public static final String ${relationship.capitalizedName}Key = "${relationship.name}";
    #else

        // Inherited relationship $relationship.name
    #end
    #end
    #foreach ($relationship in $entity.sortedClassToManyRelationships)
    #if (!$relationship.inherited)

        /**
         * Key value for to many relationship $relationship.name
         */
        public static final String ${relationship.capitalizedName}Key = "${relationship.name}";
    #else

        // Inherited relationship $relationship.name
    #end
    #end
    #if (!$entity.isAbstract)

        /**
         * Returns a newly initialized instance of ${entity.name}. The new object is inserted in the editing
        context.
         *
         * @param context
         *         editing context to insert in.
         * @return newly initialized instance
         */
        public static ${entity.prefixClassNameWithoutPackage} new${entity.classNameWithoutPackage}Instance
        (EOEditingContext context) {
            EOClassDescription description = EOClassDescription.classDescriptionForEntityName("${entity.
        name}");

            EOEnterpriseObject object = description.createInstanceWithEditingContext(context, null);
            context.insertObject(object);
            return (${entity.prefixClassNameWithoutPackage})object;
        }

        /**
         * Returns a list of all the object in the shared editing context.
         *
         * @return list of ${entity.name}
         */
        protected static NSArray<? extends ${entity.prefixClassNameWithoutPackage}> _all${entity.
        classNameWithoutPackage}Objects() {
            NSMutableArray<${entity.prefixClassNameWithoutPackage}> result = null;
            NSArray<EOEnterpriseObject> aList = ESharedEditingContext.defaultSharedEditingContext().
        objectsByEntityName().objectForKey("${entity.name}");
            if (aList != null) {
                result = new NSMutableArray<${entity.prefixClassNameWithoutPackage}>(aList.count());
                for (EOEnterpriseObject anObject : aList) {
                    result.addObject(anObject);
                }
            }
            return (result != null ? result : NSArray.<${entity.prefixClassNameWithoutPackage}> emptyArray());
        }
    #end

    /**
     * Returns the name of the entity attached to this class.
     *
     * @return name of the entity
     */
    public static String entity() {
        return "${entity.name}";
    }

    /**
     * Sole Constructor
     */
    public ${entity.prefixClassNameWithoutPackage}() {

```

```

        super();
    }
    #foreach ($fetchSpecification in $entity.sortedFetchSpecs)

        /**
         * Returns the object matching the fetch specification "$fetchSpecification.name".
         *
         * @param context
         *         editing context to fetch in.
    #foreach ($binding in $fetchSpecification.distinctBindings)
        * @param ${binding.name}Binding
        *         fetch specification binding.
    #end
        * @return list of ${entity.name}
        */
        public static NSArray<?> objectsFor${fetchSpecification.capitalizedName}(EOEditingContext context#foreach
($binding in $fetchSpecification.distinctBindings), ${binding.javaClassName} ${binding.name}Binding#end) {
            EOFetchSpecification spec = EOFetchSpecification.fetchSpecificationNamed("${fetchSpecification.name}",
"${entity.name}");
            #if (!$fetchSpecification.distinctBindings.empty)
                NSMutableDictionary<String, Object> bindings = new NSMutableDictionary<String, Object>();
            #foreach ($binding in $fetchSpecification.distinctBindings)
                if (${binding.name}Binding != null)
                    bindings.setObjectForKey(${binding.name}Binding, "${binding.name}");
            #end
                spec = spec.fetchSpecificationWithQualifierBindings(bindings);
            #end
            return context.objectsWithFetchSpecification(spec);
        }
    #end
    #foreach ($attribute in $entity.sortedClassAttributes)
    #if (!$attribute.inherited)

        /**
         * Returns the attribute "$attribute.name".
         *
         * @return $attribute.name
         */
        public ${attribute.javaClassName} ${attribute.name}() {
            return (${attribute.javaClassName})this.storedValueForKey("${attribute.name}");
        }

        /**
         * Sets the attribute "$attribute.name".
         *
         * @param value
         *         $attribute.name.
         */
        public void set${attribute.capitalizedName}(${attribute.javaClassName} value) {
            if (logger.isDebugEnabled()) {
                logger.debug( "value: " + value);
            }
            this.takeStoredValueForKey(value, "${attribute.name}");
        }

        /**
         * Initialize the attribute "$attribute.name".
         *
         * @param value
         *         $attribute.name.
         */
        public void initialize${attribute.capitalizedName}(${attribute.javaClassName} value) {
            this.set${attribute.capitalizedName}(value);
        }
    #end
    #end
    #foreach ($relationship in $entity.sortedClassToOneRelationships)
    #if (!$relationship.inherited)

        /**
         * Returns the relationship "$relationship.name".

```

```

        *
        * @return $relationship.name
        */
    public ${relationship.actualDestination.classNameWithDefault} ${relationship.name}() {
        return (${relationship.actualDestination.classNameWithDefault})this.storedValueForKey("${relationship.name}");
    }

    /**
     * Sets the relationship "$relationship.name".
     *
     * @param value
     *         $relationship.name.
     */
    public void set${relationship.capitalizedName}(${relationship.actualDestination.classNameWithDefault} value) {
        if (logger.isDebugEnabled()) {
            logger.debug("value: " + value);
        }
        this.takeStoredValueForKey(value, "${relationship.name}");
    }

    /**
     * Returns the relationship "$relationship.name".
     *
     * @return $relationship.name
     */
    public ${relationship.actualDestination.classNameWithDefault} ${relationship.name}Relationship() {
        return (${relationship.actualDestination.classNameWithDefault})this.storedValueForKey("${relationship.name}");
    }

    /**
     * Sets the relationship "$relationship.name". This method takes care of setting the reverse relationship if it exists.
     *
     * @param value
     *         $relationship.name.
     */
    public void set${relationship.capitalizedName}Relationship(${relationship.actualDestination.classNameWithDefault} value) {
        if (value == null) {
            ${relationship.actualDestination.classNameWithDefault} object = this.${relationship.name}();
            if (object != null)
                this.removeObjectFromBothSidesOfRelationshipWithKey(object, "${relationship.name}");
        } else {
            this.addObjectToBothSidesOfRelationshipWithKey(value, "${relationship.name}");
        }
    }

    /**
     * Initialize the relationship "$relationship.name".
     *
     * @param value
     *         $relationship.name.
     */
    public void initialize${relationship.capitalizedName}(${relationship.actualDestination.classNameWithDefault} value) {
        this.set${relationship.capitalizedName}Relationship(value);
    }
#end
#end
#foreach ($relationship in $entity.sortedClassToManyRelationships)
#if (!$relationship.inherited)

    /**
     * Returns the relationship "$relationship.name".
     *
     * @return list of ${relationship.actualDestination.classNameWithDefault}
     */
    @SuppressWarnings("unchecked")

```

```

public NSArray<${relationship.actualDestination.classNameWithDefault}> ${relationship.name}() {
    NSArray aList = (NSArray)this.storedValueForKey("${relationship.name}");
    if (aList != null ) {
        return aList;
    } else {
        return NSArray.emptyArray();
    }
}

/**
 * Sets the relationship "${relationship.name}".
 *
 * @param value
 *         list of ${relationship.actualDestination.classNameWithDefault}.
 */
public void set${relationship.capitalizedName}(NSArray<${relationship.actualDestination.
classNameWithDefault}> value) {
    this.takeStoredValueForKey(value, "${relationship.name}");
}

/**
 * Adds an object to the relationship "${relationship.name}". This method log a warning if the object is
not in the same editing context than the receiver.
 *
 * @param object
 *         ${relationship.actualDestination.classNameWithDefault}.
 */
@SuppressWarnings("cast")
public void addto${relationship.capitalizedName}(${relationship.actualDestination.classNameWithDefault}
object) {
    if (logger.isDebugEnabled()) {
        logger.debug( "object: " + object);
        if ( (this.editingContext() != null) && (object != null) && (!(this instanceof com.sPearWay.
access.NamedObjectInterface)) || (!(com.sPearWay.access.NamedObjectInterface)this.isReadOnly()) && (!(
(object instanceof com.sPearWay.access.NamedObjectInterface)) || (!(com.sPearWay.access.NamedObjectInterface)
object).isReadOnly()) && (! this.editingContext().equals(object.editingContext()) ) ) {
            try {
                throw new Exception("WrongEditingContext");
            } catch(Exception exception) {
                logger.warn("Relationship \"${relationship.name}\" Exception " , exception);
            }
        }
    }
    this.includeObjectIntoPropertyWithKey(object, "${relationship.name}");
}

/**
 * Removes an object from the relationship "${relationship.name}".
 *
 * @param object
 *         ${relationship.actualDestination.classNameWithDefault}.
 */
public void removeFrom${relationship.capitalizedName}(${relationship.actualDestination.
classNameWithDefault} object) {
    if (logger.isDebugEnabled()) {
        logger.debug( "object: " + object);
    }
    this.excludeObjectFromPropertyWithKey(object, "${relationship.name}");
}

/**
 * Adds an object to the relationship "${relationship.name}". This method takes care of setting the
reverse relationship if it exists.
 *
 * @param object
 *         ${relationship.actualDestination.classNameWithDefault}.
 */
public void addto${relationship.capitalizedName}Relationship(${relationship.actualDestination.
classNameWithDefault} object) {
    this.addObjectToBothSidesOfRelationshipWithKey(object, "${relationship.name}");
}

```

```

    /**
     * Removes an object from the relationship "${relationship.name}". This method takes care of setting the
     reverse relationship if it exists.
     *
     * @param object
     *         ${relationship.actualDestination.classNameWithDefault}.
     */
    public void removeFrom${relationship.capitalizedName}Relationship(${relationship.actualDestination.
classNameWithDefault} object) {
        this.removeObjectFromBothSidesOfRelationshipWithKey(object, "${relationship.name}");
    }

    /**
     * Creates and returns a new object for the relationship "${relationship.name}".
     *
     * @return ${relationship.actualDestination.classNameWithDefault}.
     */
    public ${relationship.actualDestination.classNameWithDefault} create${relationship.capitalizedName}
Relationship() {
        EOClassDescription classDescription = EOClassDescription.classDescriptionForEntityName("${relationship.
actualDestination.name}");
        EOEnterpriseObject object = classDescription.createInstanceWithEditingContext(this.editingContext(),
null);
        this.editingContext().insertObject(object);
        this.addObjectToBothSidesOfRelationshipWithKey(object, "${relationship.name}");
        return (${relationship.actualDestination.classNameWithDefault})object;
    }

    /**
     * Removes and deletes if appropriate an object from the relationship "${relationship.name}".
     * <p>
     * The delete rule for "${relationship.name}" is: ${relationship.deleteRule.name}#if (${relationship.
ownsDestination}), and own destination#end<br/>
     * Possible choices are "Nullify", "Cascade", "Deny", "NoAction"
     * </p>
     * <p>
     #if (${relationship.deleteRule.name} != "Deny")
         * The object is removed from the relationship "${relationship.name}" and its inverse relationship if it
exists.
     #if (${relationship.deleteRule.name} == "Cascade")
         * The receiver cascade the delete rule, the object will be deleted from the editing context.
     #elseif (${relationship.ownsDestination})
         * The receiver owns the object, the object will be deleted from the editing context.
     #end
     #else
         * The remove will be denied. This method does nothing.
     #end

     * </p>
     *
     * @param object
     *         ${relationship.actualDestination.classNameWithDefault}.
     */
    public void delete${relationship.capitalizedName}Relationship(${relationship.actualDestination.
classNameWithDefault} object) {
        // Delete rule for ${relationship.name} is: ${relationship.deleteRule.name}
        // Possible choices are "Nullify", "Cascade", "Deny", "NoAction"
     #if (${relationship.deleteRule.name} != "Deny")
        this.removeObjectFromBothSidesOfRelationshipWithKey(object, "${relationship.name}");
     #if (${relationship.deleteRule.name} == "Cascade")
        // Cascade delete rule
        this.editingContext().deleteObject(object);
     #elseif (${relationship.ownsDestination})
        // Own destination delete rule
        this.editingContext().deleteObject(object);
     #end
     #else
        // Deny delete rule
     #end
    }

```

```

    /**
    * Deletes all object from the relationship "$relationship.name". All delete rules are applied.
    */
    public void deleteAll${relationship.capitalizedName}Relationships() {
        for ( Iterator<${relationship.actualDestination.classNameWithDefault}> objects = this.${relationship.name}().vector().iterator(); objects.hasNext(); )
            this.delete${relationship.capitalizedName}Relationship(objects.next());
    }

    /**
    * Initialize the relationship "$relationship.name" with the list of values
    *
    * @param value
    *         list of ${relationship.actualDestination.classNameWithDefault}.
    */
    public void initialize${relationship.capitalizedName}(NSArray<${relationship.actualDestination.classNameWithDefault}> value) {
        this.set${relationship.capitalizedName}(value);
    }

    /**
    * Initialize the relationship "$relationship.name" with one object
    *
    * @param value
    *         ${relationship.actualDestination.classNameWithDefault}.
    */
    public void initialize${relationship.capitalizedName}(${relationship.actualDestination.classNameWithDefault} value) {
        this.addTo${relationship.capitalizedName}Relationship(value);
    }
}
#end
#end
}

```