Java5FeaturesAndJdo

Java5 Language Features in JDO 1.1 and 2.0

This activity clarifies the JDO spec on persistence-capable classes using the new language features in Java 5, namely enums and generics.

Generics

- generics as persistent field types ...

JDO implementations need the information about field types that in JDO 1 is provided by the metadata element collection and map. The user can specify the type of the elements of the collection and the types of the key and value of the map. For example,

```
class Employee {
...
Map<Project, Integer> projectNumbers;
Set<Skill> skillSet;
...
}
```

- generics with wildcards as persistent field types...

Generic wildcards allow the user to bound the types of persistent collection elements, or map keys and values. For example, if we know that a `Set` can only contain `Number` elements, we might declare it as a `Set<? extends Number> skills;`. Mapping this persistent field to the datastore is similar to the issue of mapping a field of a superclass to the datastore, e.g. `Number skill;`. And it is similar to mapping `Set<Number>`. We believe that as far as JDO is concerned, the implementation can consider `Set<? extends Number>` exactly as `Set<Number>`.

- fields of type identifier type ...

The only implementation class for type identifiers is Class, which cannot be persistent. I (clr) propose to wait until a use-case is developed.

- generics arrays ...

This seems to apply only to methods of generic classes and not to persistent behavior. I (clr) propose to wait until a use-case is developed.

- as persistence-capable classes ...

This usage is not well-defined. I (clr) think that most uses would involve some kind of wrapper or holder that was type-specific. I (clr) propose to wait until a use-case is developed.

Enums

Java 5 has introduced linguistic support for enumerated types in form of `enum` declarations, for example:

```
enum Season { WINTER, SPRING, SUMMER, FALL };
```

In Java, `enum` declarations have a number (surprising) features, which exceed their counterparts in other languages:

- An `enum` declaration defines a fully fledged class (dubbed an enum type).
- An enum type may have arbitrary methods and fields and may implement arbitrary interfaces.
- Enum types have efficient implementations of all the `Object` methods, are `Comparable` and `Serializable`, and the serial form is designed to withstand arbitrary changes in the enum type.

To point out commonalities, Java enum types are no different from other user-defined classes, except that

- the number of instances is fixed at compile time,
- there are no constructors that can be called,
- there's a generated method `static public T[] values()` returning an array of all instances,
- there are new, reflective methods for enums, like `Class.isEnum()` and `Class.getEnumConstants()`.

For enum type support in JDO, we have to discuss

- enum types as managed field types ...
- enum types as persistence-capable classes ...
- the new collection types `EnumSets` and `EnumMaps` ...

JDO Specific Annotations

For managed relations, we may add javax.jdo.annotation.Inverse for use in a PC class:

```
public class Department {
...
@javax.jdo.annotation.Inverse("department")
Set<Employee> employees;
...
}

public class Employee {
...
Department department;
...
}
```

This annotation may be used to generate 'mapped by' metadata.