**Java Beans**

Лет 10 назад массовое распространение получила концепция **EJB** – **Enterprise Java Beans**.

**JavaBeans** — классы в языке Java, написанные по определённым правилам. Они используются для объединения нескольких объектов в один (англ. *bean* — фасоль, кофейное зерно) для удобной передачи данных.

Спецификация Sun Microsystems определяет JavaBeans как повторно используемые программные компоненты, которыми можно управлять, используя графические конструкторы и средства IDE.

IDE – Интегрированная среда разработки.

Бизнес-логику программы представляли в виде набора высокоуровневых объектов – бинов, которые умели обмениваться сообщениями, сохранять себя, находить друг друга по имени, и еще кучу всего. Обычно это достигалось за счет специального супер-навороченного родительского класса, хотя были и другие подходы. Поведение таких объектов регламентировалось.

## **Правила описания JavaBean**

Чтобы класс мог работать как bean, он должен соответствовать определённым соглашениям об именах методов, конструкторе и поведении. Эти соглашения дают возможность создания инструментов, которые могут использовать, замещать и соединять JavaBeans.

**Правила описания гласят:**

* Класс должен иметь конструктор без параметров, с модификатором доступа public. Такой конструктор позволяет инструментам создать объект без дополнительных сложностей с параметрами.
* Свойства класса должны быть доступны через get, set и другие методы (так называемые методы доступа), которые должны подчиняться стандартному соглашению об именах. Это легко позволяет инструментам автоматически определять и обновлять содержание bean’ов. Многие инструменты даже имеют специализированные редакторы для различных типов свойств.
* Класс должен быть сериализуем. Это даёт возможность надёжно сохранять, хранить и восстанавливать состояние bean независимым от платформы и виртуальной машины способом.
* Класс должен иметь переопределенные методы equals(), hashCode() и toString().

Так как требования в основном изложены в виде соглашения, а не интерфейса, некоторые разработчики рассматривают JavaBeans, как Plain Old Java Objects, которые следуют определённым правилам именования.

Три самых известных вида EJB-бинов:

**Entity Bean** – бин, цель которого — хранить некоторые данные. В логику такого бина встроен механизм сохранения себя и своих полей в базу данных. Такой объект может быть уничтожен, а потом воссоздан из базы заново. Но кроме хранения данных у него нет никакой логики.

**Session Bean** – это функциональный бин. У каждого Session Bean есть своя функция. Один делает одно, другой другое. Такие бины работают с другими объектам и бинами, а не со своими данными.

**Session Beans** делятся на две категории.

**Stateless Session Bean** – это бин, который не хранит во внутренних переменных важных данных, нужных для его работы. Такой бин можно уничтожить, а затем заново создать, и он будет выполнять свою функцию, как и раньше.

**Statefull Session Bean** – это бин, который хранит у себя внутри данные, которые использует при работе. Если мы вызываем методы этого бина, то в каждом следующем вызове он может использовать часть данных, переданных ему в предыдущих. И все равно этот бин – это не то же самое, что обычный объект.

Но в использовании бинов тоже было не все так радужно, поэтому скоро маятник качнулся в обратную сторону. И разработчики стали все чаще использовать обычные объекты. Им даже придумали специальное название.

**POJO** (**Plain Old Java Object**) – старый обычный Java-объект. Такие объекты не обладали какими-то суперфункциями и не наследовались от суперобъектов. Самые обычные Java-объекты.

Со временем в назначении объектов/классов возникла специализация. Как результат – выделились некоторые роли, объекты которых получили новые названия.

**DTO** — Data Transfer Object – объект, который создается с целью быть использованным при транспортировке данных. Обычно к таким объектам два требования: а) уметь хранить данные, б) уметь сериализоваться. Т.е. их используют только для пересылки данных.

Создал объект, записал в него нужные данные из бизнес-логики, сериализовал в JSON/XML и отправил куда-надо. Или наоборот – пришло сообщение – де-сериализовал его в DTO-объект и вытягивай из него данные.

**Entity** – это объект, который хранится в базе данных. Но они не содержат никакой бизнес-логики. Можно сказать, что это – данные бизнес-модели.

Есть еще **DAO** – Data Access Object. Задача DAO — сохранять объекты в базу и доставать их из нее. Entity сам такой работой не занимается – он не содержит никакой логики и, следовательно, не может ничего никуда сохранять.

**Взаимодействие DAO и Entity**

UserEntity user = UserDAO.**getUserById**("1535");

**if** (user.getAge()>18)

{

user.setMobilization(true);

UserDAO.save(user);

}

**UserEntity** – это класс, который хранит данные о пользователе (User-Entity)

**UserDAO** – это класс, который достает данные (объекты **UserEntity**) из базы и сохраняет их туда, после изменений.