**1.** Обобщение позволяет указать то, что код будет работать с не конкретным интерфейсом или классом, а с некоторым не заданным типом, что придает программе ещё более общих характер. Обобщение позволяет создавать компоненты, которые могут легко использоваться с разными типами.

Одной из причин появления обобщения стало создание классов контейнеров. Контейнер предназначен для хранения объектов, используемых в программе. Контейнеры обладают большей гибкостью и отличаются по своим характеристикам от простых массивов.

**2.** Создание обобщенного класса осуществляется также, как и обычного класса, но после имени класса указывается параметризированный тип.

class Container<T>

**3.** Обобщение работает и с интерфейсами. Рассмотрим пример обобщенного интерфейса, содержащий функцию нахождения максимума:

interface Max <T extends Comparable<T>> { T max(); }

Как и при описании интерфейса, ограничим тип параметра T, указав то, что он должен реализовывать интерфейс Comporable<T>.

**4.** Чтобы определить обобщенный метод, следует указать список параметров перед возвращаемым значением.

class GenericMethod

{

<T> void printNameOfClass(T value)

{

System.out.println(value.getClass().getName());

}

}

**5.** • в обобщениях не может быть статических атрибутов обобщенного типа;

• статические методы не могут использовать значение обобщенного типа.

• можно объявлять обобщенные методы используя тип, не объявленный в классе

**6.** При использовании обобщенной реализации класса-контейнера, сперва требуется указать тип объекта, хранящегося в контейнере, в угловых скобках. В дальнейшем в контейнер можно будет помещать объекты только этого типа (или производного, так как принцип заменяемости работает и для параметризованных типов). А при извлечении вы автоматически получаете объект нужного типа.

**7.** var obj=new Class<T>();

**8.** При объявлении обобщенного класса можно указывать больше одного параметризированного типа.

Концепция нескольких объектов, «упакованных» в один объект, называется кортежем.

Получатель объекта может читать элементы, но не может добавлять их (эта концепция еще называется объектом передачи данных).

Обычно кортеж может иметь произвольную длину, а все объекты кортежа могут относиться к разным типам.

Возможно задать тип каждого объекта и при этом гарантировать, что при чтении значения будет получен правильный тип.

class Test<A, B> { A a; B b; Test(A a, B b) { this.a = a; this.b = b; } }

**9.** приведение к типу в коллекции: (type) collection

**10.** Метосимвол <?> используется при указании области используемого типа.

Например полиморфное присвоение применяется только к базовому типу, а не к параметру обобщенного типа, и когда следующее объявление является верным:

ArrayList<Animal> list = new ArrayList<Animal>();

Этот код выдаст ошибку на этапе компиляции: ArrayList<Animal> list = new ArrayList<Dog>();

В тоже время, описанние ссылки, создание объекта, и установку ссылки на созданый объект, можно представить как:

ArrayList<? extends Animal> list = new ArrayList<Dog>();

Метосимвол <?> применим как к классам так и к интерфейсам. Позволяет получить доступ к подтипу или супертипу, объявленного в аргументе типа.