Java для начинающих

Обобщения (Generic)

Обобщенное программирование

- Обобщения понадобились потому, что они позволяют писать более безопасный код, который легче читается, чем код, перегруженный переменными типа **Object** и приведениями типов. Обобщения особенно полезны для классов коллекций вроде вездесущего класса **ArrayList**.
- Обобщения похожи, по крайней мере, внешне, на шаблоны в C++. В языке C++, как и в Java, шаблоны впервые были внедрены для поддержки строго типизированных коллекций.
- Обобщенное программирование означает написание кода, который может быть неоднократно использован с объектами самых разных типов.
- Без использования дженериков в код может пробраться ошибка типов.
- Ошибка компиляции — это намного лучше, чем исключение в связи с неправильным приведением типов во время выполнения.
- Привлекательность параметров типа состоит в том, что они делают исходный код программы более удобочитаемыми безопасным.

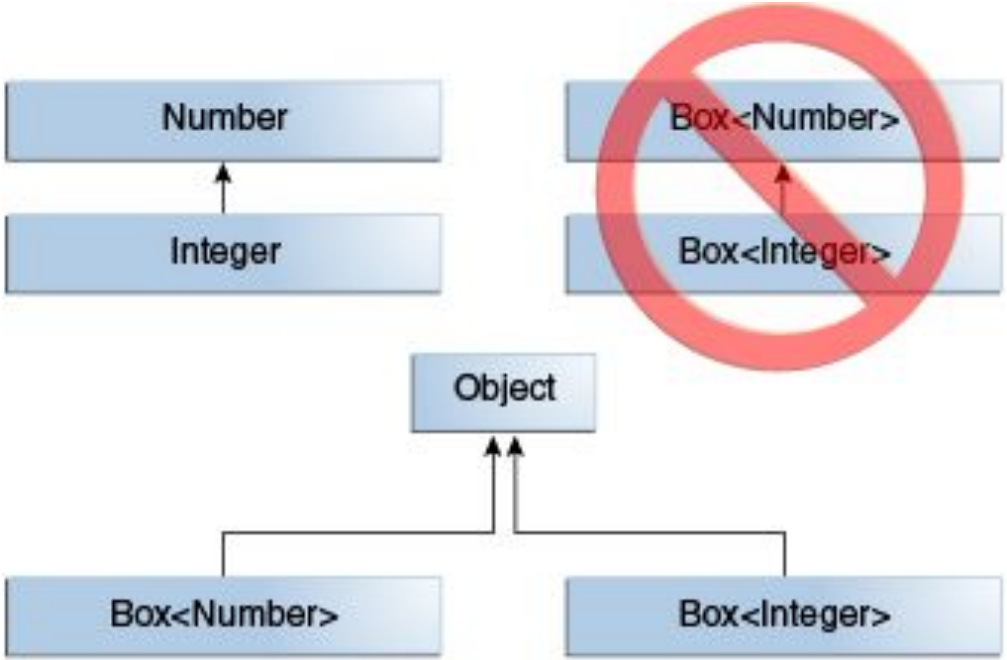
Определение обобщенного класса

- **Обобщенным** называется класс с одной или несколькими переменными типа.
- В классе вводится переменная типа **T**, заключенная в угловые скобки (**<>**) после имени самого класса.
- У обобщенного класса может быть больше одной переменной типа.
- Переменные типа используются повсюду в определении класса для обозначения типов, возвращаемых методами, а также типов полей и локальных переменных.
- Экземпляр обобщенного типа создается путем подстановки имени типа вместо переменной типа.
- Обобщенный класс действует как фабрика обычных классов

Обобщенные методы

- Метод с переменными типа:
  `public static <T> T getMiddle(T... a)`
- Переменная типа вводится после модификаторов доступа (в данном случае **public static**) и перед возвращаемым типом.
- Обобщенные методы можно определять как в обычных, так и в обобщенных классах.
- При вызове обобщенного метода, ему можно передать конкретные типы данных, заключая их в угловые скобки перед именем метода:
  `String middle = ArrayAlg.<String>getMiddle("John", "Q.", "Public");`
- При вызове метода можно пропустить параметр типа **String**. У компилятора имеется достаточно информации, чтобы вывести из такого обобщения именно тот метод, который требуется вызвать.
  `String middle = ArrayAlg.getMiddle("John", "Q.", "Public");`

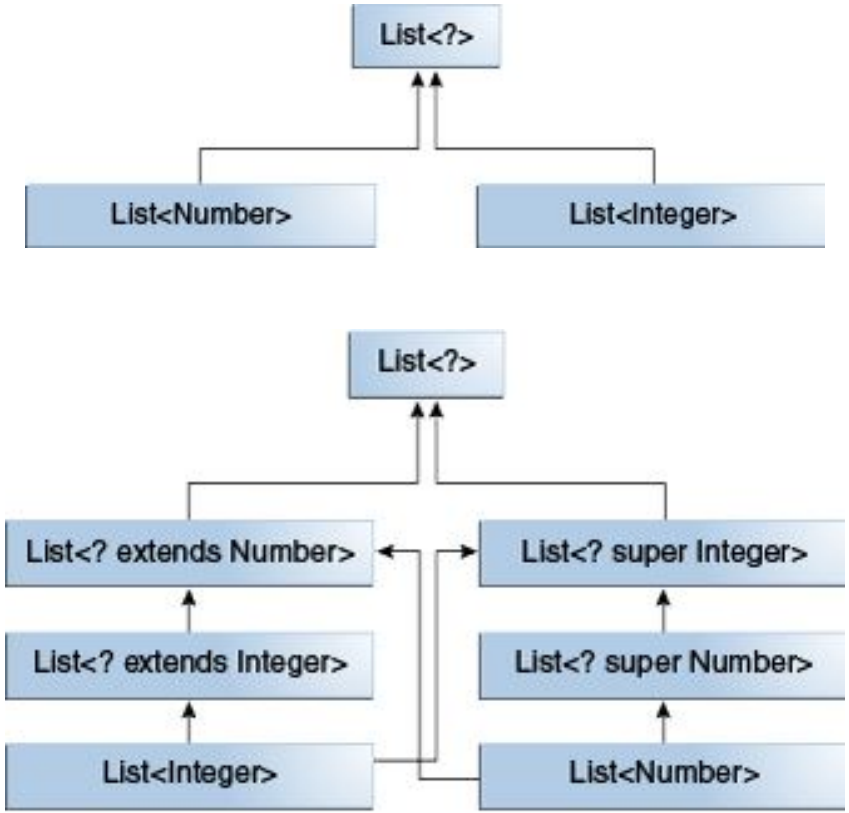
Наследование обобщений



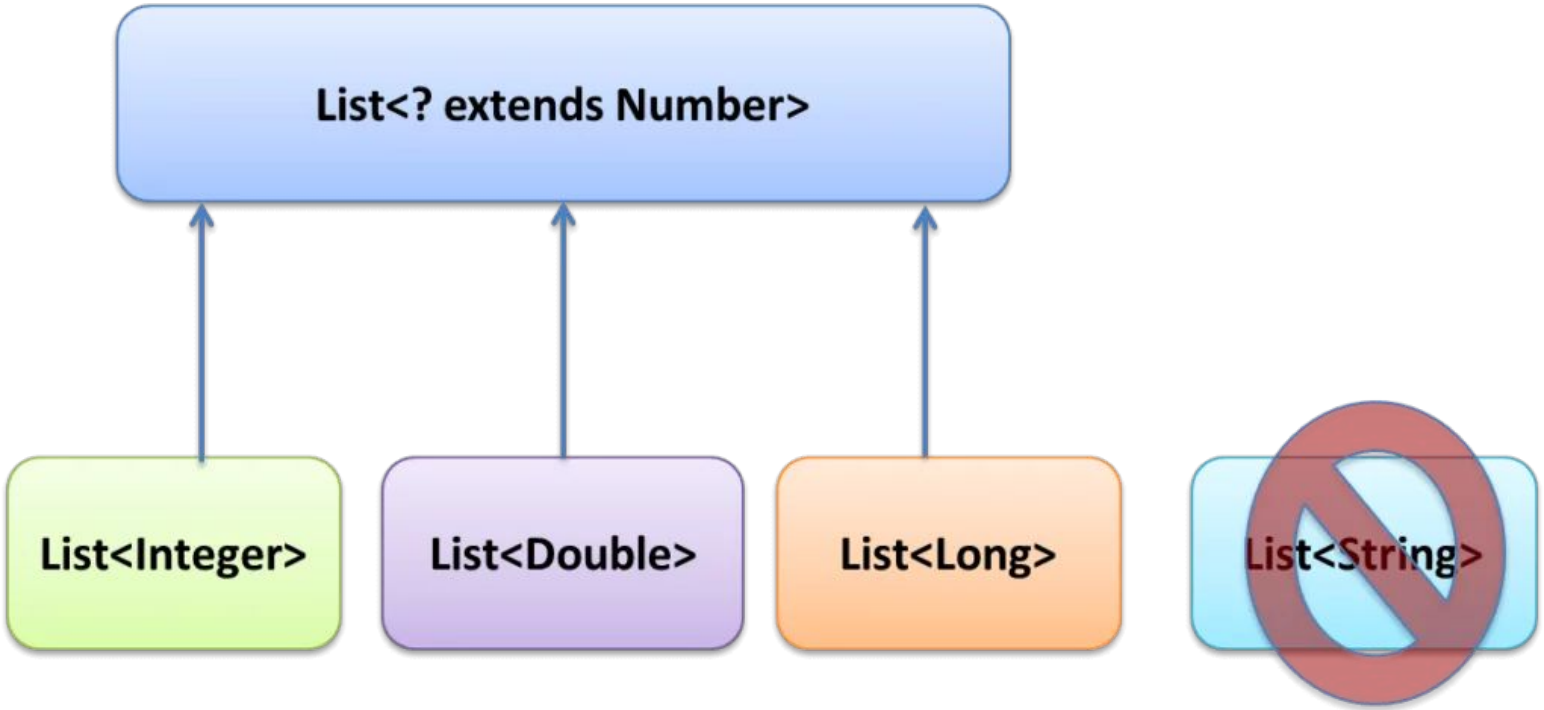
Wildcards

- Дженерики можно ограничивать верхней и нижней границей, что существенно увеличивает их мощь.
- **Wildcards** обозначаются знаком вопроса **<?>** (ещё он зовётся джокером).
- Принцип **PECS** (Producer Extends Consumer Super)
  Если метод имеет аргументы с параметризованным типом, то в случае, если аргумент - производитель (producer), нужно использовать **? extends T**, а если аргумент - потребитель (consumer), нужно использовать **? super T**.
- Если метод читает данные из аргумента, то этот аргумент - **производитель**, а если метод передает данные в аргумент, то аргумент является **потребителем**.
- Важно заметить, что определяя производителя или потребителя, мы рассматриваем только данные типа T.

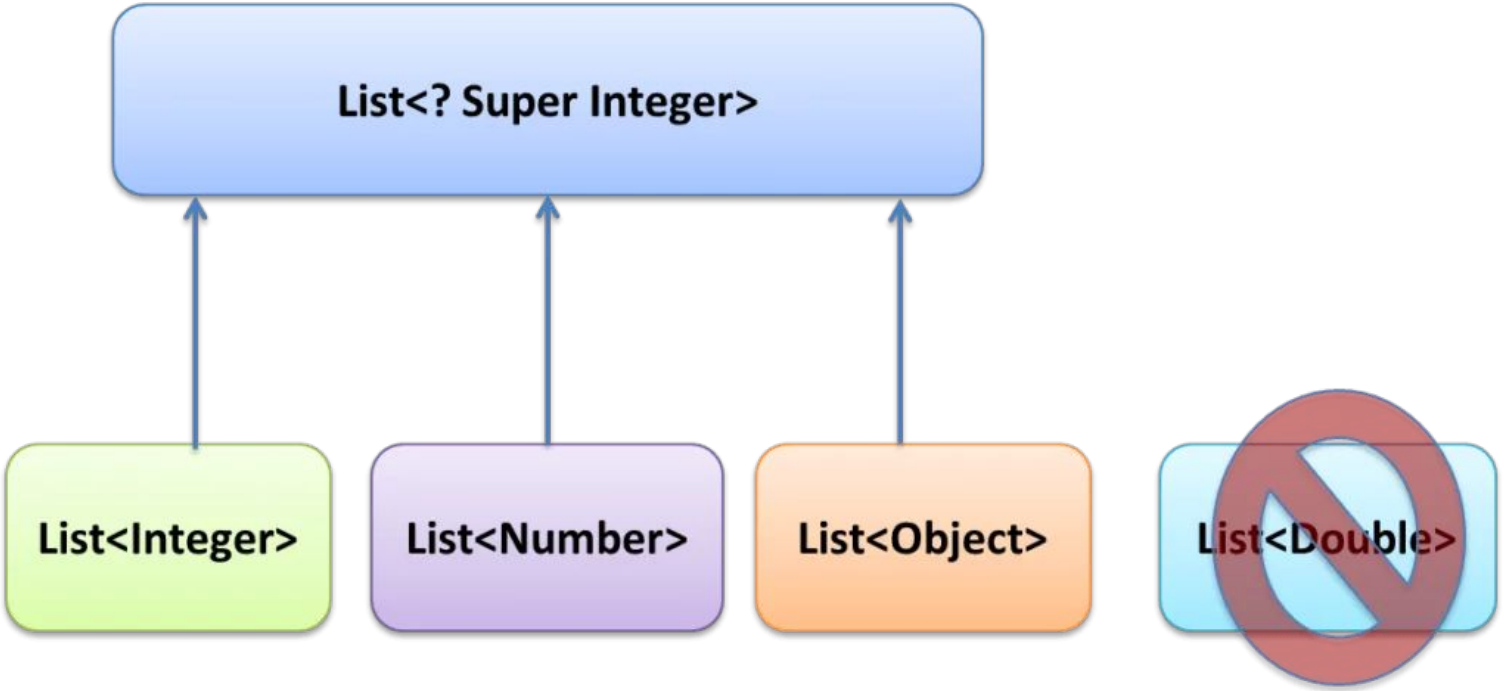
Wildcards иерархия



Wildcards иерархия



Wildcards иерархия



Ограничения на обобщения

- **Нельзя использовать с примитивными типами:**

```
Pair<int, char> p = new Pair<>(8, 'a');  // compile-time error
```

- **Нельзя создавать экземпляры параметров типа:**

```
public static <E> void append(List<E> list) {
    E elem = new E();  // compile-time error
    list.add(elem);
}
```

- **Нельзя использовать параметр типа при объявлении статического поля класса:**

```
public class MobileDevice<T> {
    private static T os;
}
```

- **Нельзя использовать оператор приведения типа `()` или оператор `instanceof`**

Ограничения на обобщения

- **Нельзя создавать массивы из параметризованных типов**

```
List<Integer>[] arrayOfLists = new List<Integer>[2];  // compile-time error
```

- **Нельзя создать, поймать (catch) или бросить (throw) объекты параметризованных типов**

```
class MathException<T> extends Exception { /* ... */ }    // compile-time error
class QueueFullException<T> extends Throwable { /* ... */} // compile-time error
```

- **Нельзя использовать при перегрузке методов**

```
public class Example {
    public void print(Set<String> strSet) { }
    public void print(Set<Integer> intSet) { }
}
```

Руководство по обобщениям

- https://docs.oracle.com/javase/tutorial/java/generics/index.html

Домашнее задание

Создать пакет **hw10**.
Реализовать параметризованный класс **NumBox**, **T** - параметр типа.
Параметром должен быть любой класс-наследник **Number** (задать необходимое условие при объявлении класса NumBox).
Класс содержит:
- массив из объектов класса T, инициализировать массив в конструкторе.
- конструктор принимающий параметр - максимальную длину массива.
- метод **void add(T num)** добавляющий число в массив. В случае если массив полон - выбросить исключение.
- метод **T get(int index)** возвращающий число по индексу.
- метод **int length()** возвращает текущее количество элементов.
- метод **double average()** - подсчет среднего арифметического среди элементов массива.
- метод **double sum()** - сумма всех элементов массива.
- метод **T max()** - максимальный элемент массива.

При подсчете воспользоваться тем, что у любого из объектов подклассов Number есть методы intValue, doubleValue, floatValue и другие.

Создать класс Main с методом main где протестировать полученный класс на примере NumBox<Float> и NumBox<Integer>.