Вопросы для подготовки:

- В чем преимущество Generic'ов, как работают? Что такое type-erasure? В чем отличие от шаблонов C++?

С помощью дженериков можно создать один класс, а потом на основе него — несколько объектов этого класса для разных типов. Не понадобится дублировать код и усложнять программу. Многие называют дженерики "синтаксическим сахаром". И они правы, так как дженерики действительно при компиляции станут теми самыми кастами.

После компиляции какая-либо информация о дженериках стирается (вместо них – касты). Это называется "Стирание типов" или "Type Erasure".

Стирание типов и дженерики сделаны так, чтобы обеспечить обратную совместимость со старыми версиями JDK, но при этом дать возможность помогать компилятору с определением типа в новых версиях Java.

Шаблоны C++ могут использовать примитивные типы, как, например, int, а дженерики Java — нет, они обязаны использовать Integer.

Java позволяет указывать ограничения на тип, передаваемый в качестве параметра. Например, вы можете использовать дженерики для реализации CardDeck и указать, что параметр типа должен наследоваться от CardGame.

В C++ можно создать экземпляр типа, передаваемого параметром, а Java — нет.

Java не позволяет использовать типы, передаваемые параметром (например, Foo в MyClass<Foo>) для статических методов и переменных, так как они могут совместно использоваться в MyClass<Foo> и MyClass<Bar>. В C++ — это разные классы, поэтому тип из параметра можно использовать для статических методов и переменных.

В Java все экземпляры MyClass<T>, независимо от их параметров, относятся к одному и тому же типу. Параметры типов уничтожаются после компиляции. В C++ экземпляры с разными параметрами типов — различные типы.

- Расскажите про Wildcard

**public** **static** **void** iterateAnimals(Collection<? **extends** Animal> animals) {

**for**(Animal animal: animals) {

System.out.println("Еще один шаг в цикле пройден!");

}

}

Это и есть wildcard. Точнее, это первый из нескольких типов wildcard — “**extends**” (другое название — **Upper Bounded Wildcards**). О чем нам говорит эта конструкция? Это значит, что метод принимает на вход коллекцию объектов класса Animal либо объектов любого класса-наследника Animal (? extends Animal).

**public** **static** **void** iterateAnimals(Collection<? **super** Cat> animals) {

**for**(**int** i = 0; i < animals.size(); i++) {

System.out.println("Еще один шаг в цикле пройден!");

}

}

Здесь принцип похож. Конструкция <? super Cat> говорит компилятору, что метод iterateAnimals() может принимать на вход коллекцию объектов класса Cat либо любого другого класса-предка Cat.

- Что такое ковариантность/контравариантность?

Формально, ковариантность/контравариантность типов – это сохранение/обращение порядка наследования для производных типов. Проще говоря, когда у ковариантных сущностей типами-параметрами являются родитель и наследник, они сами становятся как бы родителем и наследником. Контравариантные наоборот, становятся наследником и родителем.

Легче всего осознать эти понятия на примерах:

🔘 Ковариантность: List<Integer> можно присвоить в переменную типа List<? extends Number> (как будто он наследник List<Number>).

🔘 Контравариантность: в качестве параметра метода List<Number>#sort типа Comparator<? super Number> может быть передан Comparator<Object> (как будто он родитель Comparator<Number>)

Отношение типов «можно присвоить» – не совсем наследование, такие типы называются совместимыми (отношение «is a»).

Существует еще одно связанное понятие – инвариантность. Инвариантность – это отсутствие свойств ковариантности и контрвариантности. Дженерики без вайлдкардов инвариантны: List<Number> нельзя положить ни в переменную типа List<Double>, ни в List<Object>.

Массивы ковариантны: в переменную Object[] можно присвоить значение типа String[].

- Что означает принцип PECS?

PECS (Producer Extends Consumer Super) - это принцип, который используется при работе с generics в Java. Он определяет, какие типы wildcards ("?" символ) следует использовать для обобщенных типов данных, когда мы работаем с производителями (producer) и потребителями (consumer).

Производители - это объекты, которые генерируют значения типа T. Например, если у нас есть список фруктов, то производителем будет метод, который возвращает элементы списка.

Потребители - это объекты, которые используют значения типа T. Например, если у нас есть список фруктов, то потребителем может быть метод, который выводит элементы списка на экран или сохраняет их в файл.

Согласно принципу PECS, если мы хотим использовать обобщенный тип данных, как производитель, то следует использовать wildcard "extends", а если мы хотим использовать его в качестве потребителя, то следует использовать wildcard "super".

- Что такое Bounded Wildcard?

- Что такое Generic метод?

Дженерики-классы и дженерики-методы

Выше мы говорили, что дженериками могут быть разные сущности. Разберемся подробнее:

дженерик-классы (generic classes) — это классы, «схемы» объектов с параметром. При создании объекта ему передается тип, с которым он будет работать;

дженерик-методы (generics methods) — это методы, работающие по такому же принципу. Метод — это функция внутри объекта, то, что он может делать. Методу тип передается при вызове, сразу перед аргументами. Так можно создавать более универсальные функции и применять одну и ту же логику к данным разного типа.

- Как Generics используется в классах?

С помощью дженериков можно создать один класс, а потом на основе него — несколько объектов этого класса для разных типов

- Что вы знаете про Bounded type argument (проблема несовместимости generic-типов)?

Как говорится: "В бочке мёда есть ложка дегтя". Для того чтобы сохранить целостности и независимости друг от друга Коллекции, у Generics существует так называемая "Несовместимость generic-типов".

* **Пример**:

List<Integer> li = new ArrayList<Integer>();

List<Object> lo = li;

Иначе — ошибки

lo.add(“hello”);

// **ClassCastException**: String -> int

Integer li = lo.get(0);

- Что такое Lower bounded wildcard?

The Upper Bounded Wildcards section shows that an upper bounded wildcard restricts the unknown type to be a specific type or a subtype of that type and is represented using the extends keyword. In a similar way, a lower bounded wildcard restricts the unknown type to be a specific type or a super type of that type.

(В разделе "Подстановочные знаки с верхней границей" показано, что подстановочный знак с верхней границей ограничивает неизвестный тип конкретным типом или подтипом этого типа и представляется с помощью ключевого слова extends. Аналогично, нижний подстановочный знак ограничивает неизвестный тип конкретным типом или super типом этого типа.)

- Что такое Wildcard Capture (или "Capture helpers")?

В некоторых случаях компилятор сам определяет тип подстановочного знака. Например, список может быть определен как List, но при вычислении выражения компилятор выводит из кода конкретный тип. Такой сценарий известен как захват подстановочного знака.  
  
По большей части вам не нужно беспокоиться о захвате подстановочных знаков, за исключением тех случаев, когда вы видите сообщение об ошибке, содержащее фразу "capture of".  
  
Пример WildcardError при компиляции выдает ошибку захвата:  
В этом примере компилятор обрабатывает входной параметр i как имеющий тип Object. Когда метод foo вызывает List.set(int, E), компилятор не может подтвердить тип объекта, который вставляется в список, и возникает ошибка. Когда возникает подобная ошибка, это, как правило, означает, что компилятор считает, что переменной присвоен неправильный тип. Именно для этого в язык Java были добавлены генерики - для обеспечения безопасности типов во время компиляции.  
  
Пример WildcardError при компиляции в реализации Oracle JDK 7 javac выдает следующую ошибку:  
  
В данном примере код пытается выполнить безопасную операцию, так как же можно обойти ошибку компилятора? Ее можно устранить, написав частный метод-помощник, который перехватывает подстановочный знак. В данном случае проблему можно обойти, создав частный метод-помощник fooHelper, как показано в WildcardFixed:  
public class WildcardFixed {

void foo(List<?> i) {

fooHelper(i);

}

// Helper method created so that the wildcard can be captured

// through type inference.

private <T> void fooHelper(List<T> l) {

l.set(0, l.get(0));

}

}  
- Что вы знаете про наследование исключений в generic'ах?

- Метод принимает ссылку на List<Parent>. Child наследуется от Parent. Можно ли в метод передать List<Child>?

В типе аргумента нужно указать List<? extends Parent>

- Есть ли отличия в записи List<?> и List<? extends Object>?

Все классы без исключения наследуются от Object. Поэтому неограниченный wildcard <?> всегда подразумевает его в качестве верхней границы. Оба этих типа в рантайме сотрутся в List<Object>, функциональных отличий нет.