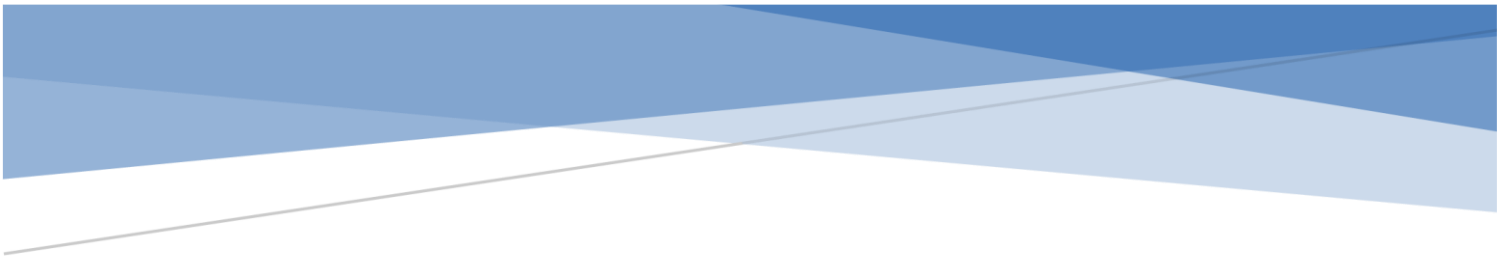

INHERITANCE

Checklist

Olga Smolyakova

Ответьте на вопросы, приведенные ниже. Нарисуйте рисунки, позволяющие вам объяснить ответы.

Напишите (нарисуйте) алгоритм, описывающий порядок вызова конструкторов при создании объектов производного класса. (Постарайтесь предусмотреть все случаи)	При создании объекта в первую очередь вызывается конструктор его базового класса, а только потом — конструктор самого класса, объект которого мы создаем. Поэтому слово super() всегда должно стоять в конструкторе производного класса первым, иначе логика работы конструкторов будет нарушена и программа выдаст ошибку. Если конструктор производного класса явно не вызывает конструктор базового класса, то происходит вызов конструктора по умолчанию базового класса. В этом случае в базовом классе должен быть определен конструктор по умолчанию.

Как вы думаете, зачем необходимо переопределение методов? Приведите примеры переопределенных методов. Может ли при переопределении метода меняться тип возвращаемого значения?	Переопределение методов позволяет подклассу обеспечивать специфическую реализацию метода, уже реализованного в суперклассе.
	У переопределенного метода должен быть тот же тип возвращаемого значения, что и у метода родителя, в ином случае мы получим ошибку компиляции.
	
	`class BaseClass {`
	`  public void doSmth() {`
	`  System.out.println("Do some basic things");`
	`  }`
	`}`
	
	`class SubClass extends BaseClass {`
	`  public void doSmth() {`
	`  System.out.println("Do some sub things");`
	`  }`
	`}`

Объясните, как вы понимаете смысл фразы “ссылка базового типа может ссылаться на объекты производных типов” и “объект подкласса может быть использован везде, где используется объект его суперкласса.”	Так как производный класс наследуется от базового класса, то объект производного класса по сути является в то же время и объектом базового класса. Поэтому ссылка базового класса может ссылаться на объекты производных типов. Здесь также важно понятие “динамическое связывание”. При динамическом связывании правильный метод будет вызываться только во время выполнения, после того как программа узнает, какой именно будет объект.
Попробуйте порассуждать на тему, наследуются или нет статические методы.	Статические методы используются для функциональности, принадлежат классу «в целом», а не относятся к конкретному объекту класса. Статические методы в Java наследуются, но не могут быть переопределены. Если мы попытаемся переопределить статические методы, они просто скроют статические методы суперкласса, а не переопределят их. Статические методы не являются полиморфными. Во время компиляции статический метод будет статически связан.

Как вы понимаете, то такое перегруженные методы? Приведите примеры перегруженных методов.	Перегрузка методов - это возможность создавать несколько методов с одинаковым названием, но разными параметрами (с разной сигнатурой) и возвращаемыми значениями. Статические методы могут перегружаться нестатическими и наоборот.
	
	class TestClass {
	public void doSmth() {
	System.out.println("Do something");
	}
	public int doSmth(int a) {
	return a+1;
	}
	public int doSmth(int a, int b) {
	return a+b;
	}
	}

Как вы думаете, зачем в языке Java надо применять ключевое слово final. Что может быть final в Java.	С помощью ключевого слова final можно предотвратить переопределение методов - оно делает эти методы терминальными. Если же сделать класс терминальным, то его нельзя будет расширить (т.е. не может иметь подклассов). Также final может применяться к переменным. В этом случае однажды присвоенное значение переменной не может быть изменено. Для ссылочных переменных это означает, что после присвоения объекта, нельзя изменить ссылку на данный объект. Ссылку изменить нельзя, но состояние объекта изменять можно.
Перечислите методы класса Object, которые вы знаете. Расскажите для чего они предназначены и как работают.	public String toString() - возвращает строковое представление объекта. По-умолчанию возвращает имя_класса@hashcode в 16-ричной системе. Если hashcode не переопределен, то вернет значение по-умолчанию. public native int hashCode() - используются для сравнения объектов. Необходимо переопределять, если был переопределен equals. public boolean equals(Object obj) - используются для сравнения объектов. Для этого его необходимо переопределить в подклассе.
Для чего следует применять метод equals(). Знаете ли вы "где" в Java метод equals() применяется неявно? Перечислите правила переопределения метода equals()	Метод equals() необходим для сравнения двух объектов (нужно переопределить в соответствии с правилами). Вероятно, метод equals() применяется неявно в HashMap. Правила при переопределении equals: Рефлексивность: x.equals(x) == true Симметричность: если x.equals(y), тогда y.equals(x) == true Переносимость: если x.equals(y) и y.equals(z), тогда x.equals(z) == true Консистентность: если x.equals(y), тогда x.equals(y) == true Null проверка: x.equals(null) == false

Что такое хэш-код? Что такое хэш-код объекта? Объясните, почему хэш-коды двух различных объектов могут совпасть. Перечислите известные вам правила переопределения метода hashCode()	Хэш-код - это число. Если более точно, то это битовая строка фиксированной длины, полученная из массива произвольной длины с помощью определенного алгоритма. Функция hashCode() возвращает для любого объекта 32-битное число типа int. Значение по-умолчанию - целочисленный адрес в памяти. По хорошему, у разных объектов хешкод должен быть разный. Но на практике иногда происходит по другому. Очень часто это происходит из-за несовершенства формулы для вычисления хешкода. Т.к. возвращаемый тип метода hashCode() - int, у int есть определенный предел. Если разных объектов будет больше, чем этот предел, то физически нельзя сгенерировать разные хеши для всех объектов.	Во время работы приложения значение хэш-кода объекта не изменяется, если объект не был изменен. Все одинаковые по содержанию объекты одного типа должны иметь одинаковые хэш-коды. Различные по содержанию объекты одного типа могут иметь различные хэш-коды.
Расскажите, когда применяется метод toStirng() и как его необходимо переопределять	Этот метод позволяет получить текстовое описание любого объекта. Реализация его в классе Object следующая: return getClass().getName() + "@" + Integer.toHexString(hashCode()); Данный метод можно переопределить в любом классе и возвращать более нужное или более детальное описание объекта. При создании нового класса принято переопределение toString() таким образом, чтобы возвращающая строка содержала в себе имя класса, имена и значения всех переменных.	