Создание класса – наследника. Часть 1. Основные действия.

Для создания класса наследника воспользуемся уже созданным в предыдущем разделе классом Point() ( см. пример 02\_InitializationClass\_01 ).

1. Создаем новый JAVA-файл Circle.java для создания в нем класса-наследника Circle():

**public class** Circle **extends** Point{}.

1. Описываем данные:

**private double radius**;

координаты не требуется описывать, т.к. они уже реализованы в базовом классе.

1. С помощью команды Generate… (ПКМ) или самостоятельно реализуем конструктор (необходимое количество). Следует отметить отсутствие конструктора по умолчанию (круг с радиусом 0 – точка). Соответственно реализуются 2 конструктора – основной:

**public** Circle(**double** x, **double** y, **double** radius)

и видоизмененный – где устанавливается только радиус:

**public** Circle(**double** radius)

(координаты устанавливаются по умолчанию).

1. Соответственно, учитывая, что радиус не может быть 0 и меньше нуля – реализуем класс:

**public final void** checkRadius(**double** radius) **throws** IllegalArgumentException

который проверяет радиус на соответствие условиям и в случае передачи в класс отрицательного или нулевого значения (ошибочных данных) генерирует соответствующее сообщение об ошибке. Данное сообщение об ошибке из серии непроверяемых (unchecked). Кроме генерации сообщения об ошибке, рекомендуется сообщить о возможной генерации ошибок в заглавии метода. Это необязательное действие, но это рекомендуется делать – т.к. экземпляр класса не будет создан или будет создан некорректно.

Ремарка: Метод checkRadius получил модификаторы **public final** , которые с одной стороны предоставляют доступ к этому методу в том числе и от наследников данного класса (**public**), с другой же стороны запрещают его модификации (**final**). В случае, если такой доступ не желателен, то достаточно данному методу предоставить доступ **private**.

Ремарка 2: Если же создание экземпляра класса критично – рекомендуется генерировать проверяемую (checked) ошибку с указанием о ней в заглавии метода (через **throws** с соответствующим типом ошибки). Такое действие вынудит обрабатывать такую ситуацию.

Ремарка 3: Генерация ошибки при создании (изменении состояния) объекта не является хорошей практикой. Поэтому рекомендуется в соответствующем контроллере совершать проверку всех данных на соответствие ограничениям и только после этого вызывать конструктор или метод, который изменяет состояние объекта.

1. С помощью команды Generate… (ПКМ – Override Methods) реализуем (переопределяем) основные методы ( **setAllParameters(double x, double y) и moveCoordinate(double dX, double dY)** ). При необходимости дописуем их реализацию. Следует отметить, что в случае изменения аргументов метода (что произошло в методе **setAllParameters(double x, double y)**) аннотацию @Override не следует применять, т.к. метод переопределен (overload), а не расширен(override).
2. С помощью команды Generate… (ПКМ) или самостоятельно реализуем необходимые геттеры и сеттеры.
3. С помощью команды Generate… (ПКМ) или самостоятельно (при необходимости) переопределяем метод toString().