Инициализация класса. Часть 3. Метод clone()

Для демонстрации особенностей метода clone() воспользуемся уже созданным в предыдущем разделе классом Point()( см. пример 04\_Equals&HashCode\_02 ).

1. Прописываем в заглавии класса Point() имплементацию интерфейса Cloneable :

**public class** Point **implements** Cloneable

1. С помощью команды Generate… (ПКМ – Override Metods… - clone():Object) реализуем (переопределяем) этот метод:

**protected** Object clone() **throws** CloneNotSupportedException {  
 **return super**.clone();  
}

1. Создаем объект и пробуем создать его клон:

Point p1 = **new** Point(3,4);  
Point p3 = p1.clone();

Как мы видим, т.к. исключение (Exception) обрабатываемое (checked), и от нас требуют его обработать (в реальных программах обработка в момент клонирования объекта в месте вызова метода clone() является предпочтительной). Однако, чтобы не отвлекаться на постоянную обработку исключения при демонстрации особенностей метода clone(), перепишем его таким образом, что данное исключение будет обработано внутри метода:

**public** Point clone() {  
 **try**{  
 *//return super.clone();* Point temp = (Point)**super**.clone();  
 **return** temp;  
 }**catch** ( CloneNotSupportedException ex) {  
 ex.printStackTrace();  
 **return null**;  
 }  
}

1. После изменений в методе clone() создаем клон и проверяем это:

System.out.println(p1 == p2);  
System.out.println(p1.equals(p2));  
System.out.println(p1.hashCode());  
System.out.println(p2.hashCode());  
System.out.println(p1.hashCode() == p2.hashCode() );

Результат выполнения программы:

false

true

17301504

17301504

true

доказывает нам, что мы имеем два идентичных объекта.

1. Для того, чтобы окончательно удостовериться в том, что у нас два несвязанных друг с другом объекта – изменим координату одного:

p1.setX(6);

и опять сравним.

false

false

49807360

17301504

False

3.0 (System.***out***.println(p2.getX()); )

( см. пример 05\_Clone ).

1. Теперь посмотрим, справедливо ли это и для ссылочных типов. Для этого введем в класс Point() ссылочный тип

**public void** setHistory(ArrayList<Point> history) {  
 **this**.**history** = history;  
}

в который будем заносить старые значения координат при их изменениях. Изменим методы в соответствии с поставленной задачей и реализуем гетеры и сетеры, hashCode() и equals().

1. Повторяем процедуру клонирование объекта и изменяем данные в ArrayList<Point> history базового объекта:

p1.getHistory().add(**new** Point(5,6));

Т.к. клонированный объект не претерпел изменений, то он должен отличаться от базового.

Проверяем на связанность исходного объекта с клоном:

false

true

33816576

33816576

True

как мы видим – изменения в ссылочном типе первого объекта отобразились и в ссылочном объекте клона. Соответственно, можно увидеть, что исходный объект остался связанным со своим клоном по ссылочным полям.

( см. пример 05\_Clone\_02 ).

1. Т.е. можно сделать вывод, что при наличии в объектах ссылочных типов стандартной реализации метода clone() (переопределение метода clone() средствами IDE) недостаточно. Поэтому внесем явное клонирование ссылочного типа (ArrayList<Point> history) в метод clone():

*//reference type MUST HAVE be cloned*temp.**history** = (ArrayList<Point>)**this**.**history**.clone();

1. Проверяем на связанность исходного объекта с клоном:

[Point{x=3.0, y=4.0}, Point{x=5.0, y=6.0}]

[Point{x=3.0, y=4.0}]

false

false

1803551713

1584660512

false

и видим, что связанность на уровне ArrayList<Point> history исчезла.

( см. пример 05\_Clone\_03 ).

1. Однако у нас внутри ArrayList<Point> history находятся объекты типа Point(). Проверим, нет ли лишних связей на этом уровне между исходным объектом и клоном. Для этого изменим значение объекта Point() в исходном объекте и посмотрим, что произойдет в клоне:

p1.getHistory().get(0).setX(100500);  
System.***out***.println(p1.getHistory());  
System.***out***.println(p2.getHistory());

и видим, что при изменении в исходном объекте значения в объекте Point()из ArrayList<Point> клона отображаются аналогичные изменения:

[Point{x=100500.0, y=4.0}]

[Point{x=100500.0, y=4.0}]

false

true

-446219936

-446219936

true

Соответственно внесенных изменений на уровне ArrayList<Point> недостаточно и приходится внести метод clone() еще и явное клонирование объектов Point()из ArrayList<Point>:

*// objects by references in arrays can be cloned*temp.**history** = **new** ArrayList<>();  
**for**(Point p : **this**.**history** ){  
 temp.**history**.add((Point) p.clone());  
}

1. Проверяем:

[Point{x=100500.0, y=4.0}]

[Point{x=3.0, y=4.0}]

false

false

-446219936

1584660512

false

и видим, что исходный объект и его клон стали полностью независимы.

( см. пример 05\_Clone\_03 ).

1. Подведем итог: В случае если в исходном объекте присутствуют ссылочные типы данных стандартной реализации метода clone() (переопределение метода clone() средствами IDE через Generate… (ПКМ – Override Metods… - clone():Object)) НЕДОСТАТОЧНО.

При этом разработчик должен быть очень внимателен, т.к. внутри ссылочного типа данных могут также находиться ссылочные типы данных. И в этом случае если эти типы данных не будут явно клонированы в методе clone() в дальнейшем могут возникнуть трудно выявляемые баги.