Lab work 4

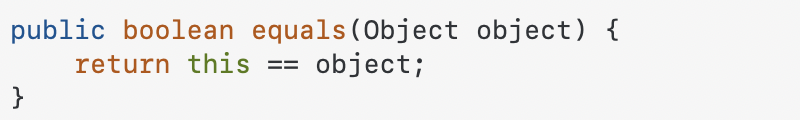
Java *equals()* and *hashCode()*

В этом руководстве мы познакомимся с двумя тесно связанными между собой методами: *equals()* и *hashCode()*. Мы сосредоточимся на их отношении друг к другу, на том, как правильно их переопределять и почему мы должны переопределять либо оба этих метода либо ни один из них.

***equals()***

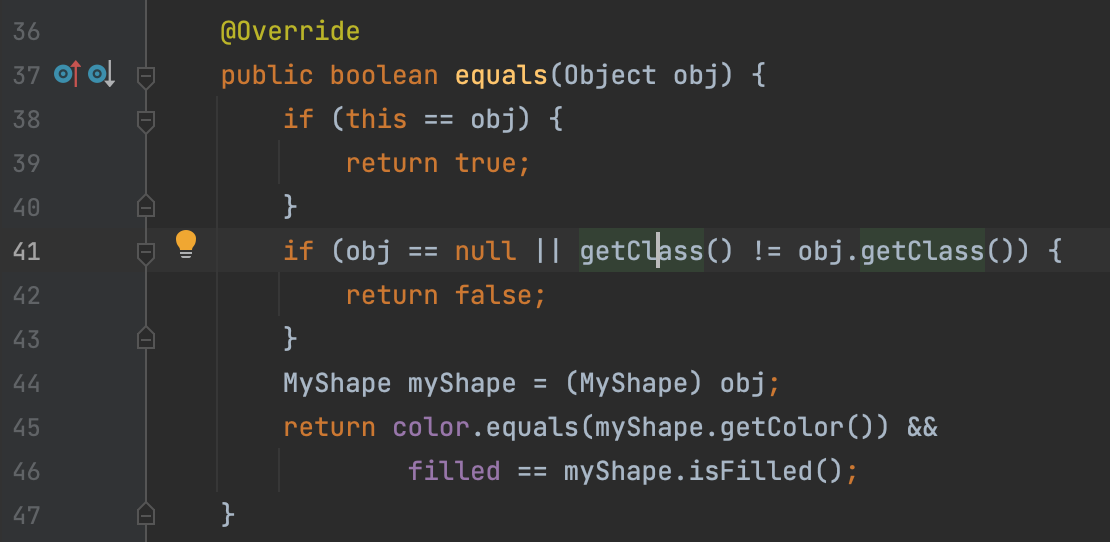
В классе *Object*,от которого наследуются все классы в Java, уже описаны методы *equals()* и *hashCode()*, т.е. любой класс что мы создаем уже имеет эти два метода унаследованные от класса *Object*.

Метод *equals()* в классе Object сверяет объекты по их адресу (оператор *==* работает только с примитивами, если же с помощью этого оператора сверяют объекты, то идет сверка адресов в памяти)



поэтому если не переопределять у себя в классе метод *equals()*, то даже создав два идентичных объекта этого класса, мы никогда не получим *equals()* == true, так как эти два объекта имеют разные адреса в ячейках памяти.

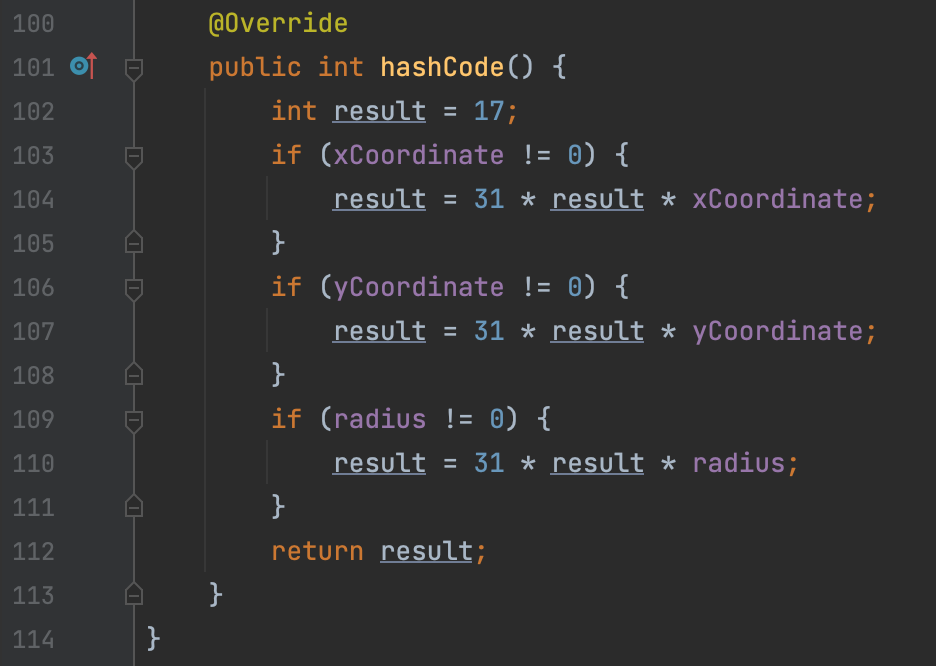
Пример переопределения метода *equals()* в классе *MyShape*:



Как вы видите мы написали свою реализацию метода сравнения с другим объектом в которой сравниваем в первую очередь адреса в ячейках памяти (оператор ==), если адреса совпадают то это значит это один и тот-же объект (возвращаем true), затем сравниваем классы объектов (метод *getClass()* возвращает класс объекта), если у объектов разные классы, то это точно разные объекты. На строке 44 мы явно указываем что мы обращаемся к объекту *obj* как к объекту класса *MyShape* (*(MyShape) obj*) и создаем новый объект на его основе (*myShape*). Последним делом мы сравниваем поля этого объекта с полями нашего объекта, если они совпадают то возвращаем *true*, если же нет возвращаем *false*. Таким способом мы написали метод с помощью которого сможем сравнивать объекты нашего класса *MyShape*.

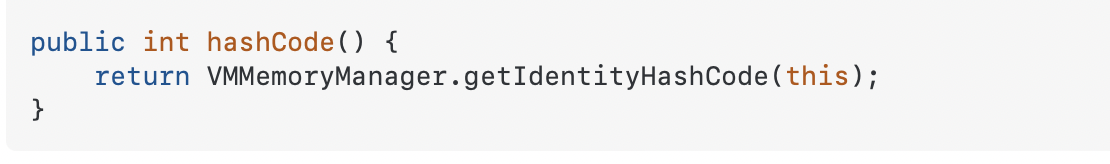
**Задание номер 1**: Переопределить метод *equals()* во всех классах созданных в прошлых лабораторных работах (*MyShape, Circle, Rectangle, Square*) по примеру указанному выше.

Пример *hashCode()* у класса *Circle*.



***hashCode()***

Метод *hashCode()* возвращает 32 битное значение типа int. В Java этот метод реализован на языке C (native метод, т.е. написанный на другом языке и вызванный из Java кода). Он также описан в классе *Object*:



Мы не будем углубляться в детали реализации этого метода на C. Нам следует знать что если мы хотим сравнивать наши объекты и переписываем метод *equals()* то мы обязаны переписать и метод *hashCode()* так как есть правило работы этих двух методов:

1. Если *x.equals(y) == true*, то обязательно *hashCode(x) == hashCode(y)*

При этом - если *hashCode(x) == hashCode(y)*, то не обязательно *x.equals(y) == true*, т.е. у двух разных объектов могут быть одинаковые хэш значения (зависит от реализации метода *hashCode()*).

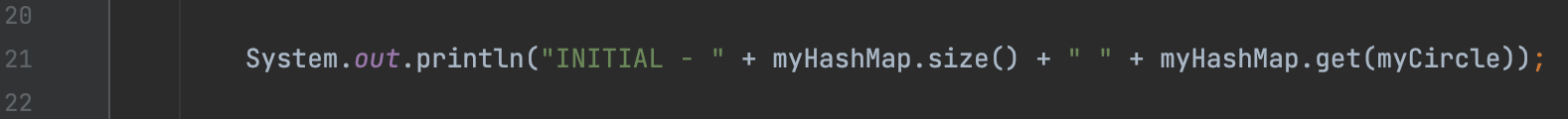
Метод *hashCode()* должен возвращать хэш объекта с помощью которого с ним будут работать другие объекты в Java (такие как *HashMap*, *HashSet* и т.д.)

К примеру *HashMap* - это хранилище типа ключ-значение, вы можете создавать элементы и можете хранить их в *HashMap*, затем доставать, изменять и удалять их из нее.

Пример использования *HashMap* с нашими объектами наследниками класса *MyShape*:



*<MyShape, String>* означает что ключом является класс *MyShape* а значением является строка (*String*), мы может сохранять наши ключ-значения в мапу и позже доставать их из нее:



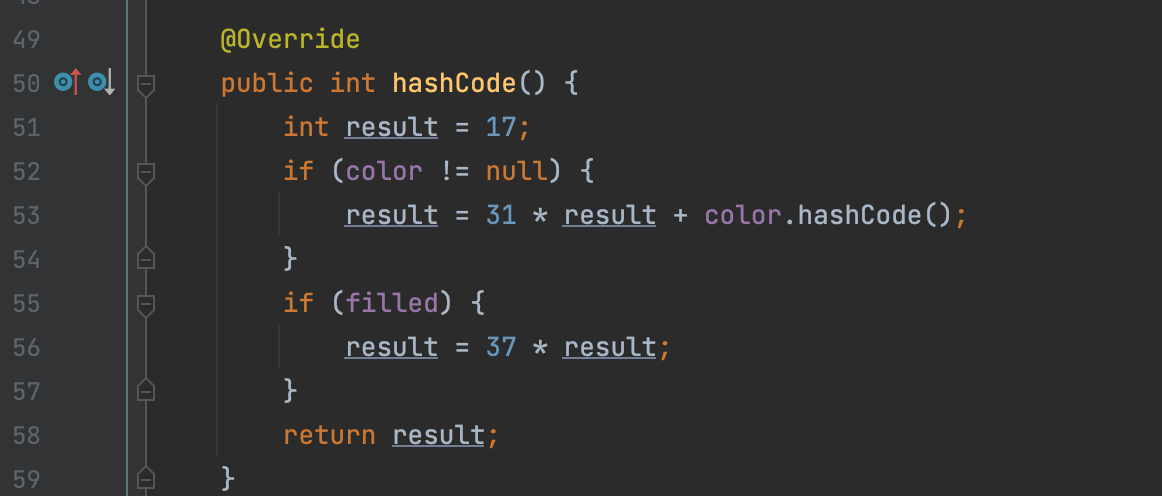
Эта команда выведет нам следующее:



То есть мы по нашему объекту (ключу) *myCircle* достали значение “CIRCLES” добавленное раннее.

Но чтобы *HashMap* корректно работал с нашими объектами (то есть правильно искал по ключам) нам нужно чтобы были правильно описаны методы *hashCode()* (вычислять хэш по которому будет искаться ключ внутри *HashMap*) и *equals()* (будет использоваться в сравнении ключей если в *HashMap* уже есть объекты с таким хэшем, как вы помните у разных объектов **могут** быть одинаковые хеши, поэтому такие объекты будут дополнительно сверяться методом *equals()*). Если же мы будем использовать дефолтный *hashCode()* от класса *Object*, то у всех объектов (даже одинаковых) будут разные хэши и у нас будут одинаковые ключи в *HashMap*, и наоборот если все объекты будут возвращать одинаковый хэш (константный ответ для всех вызовов) то наш *HashMap* будет работать медленнее так как по одному и тому же хэшу будут находиться все объекты.

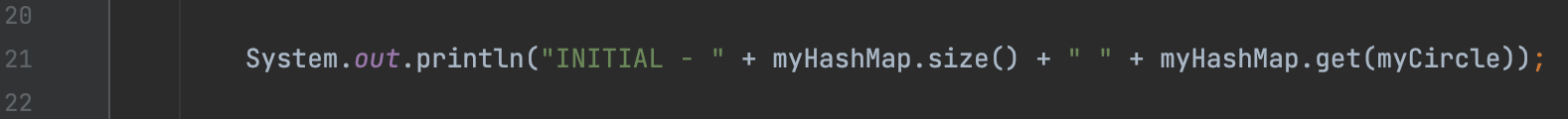
Пример реализации метода *hashCode()* в классе *MyShape*:



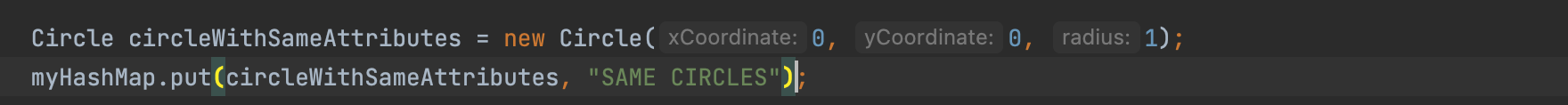
**Задание номер 2:** Переопределить метод *hashCode()* во всех классах созданных в прошлых лабораторных работах (*MyShape, Circle, Rectangle, Square*) по примеру указанному выше.

**Задание номер 3:** Заполнить HashMap как в примере 

Вывести на экран количество элементов и значение по ключу одного из объектов



Создать новый элемент (с такими же параметрами как и один из уже загруженных в *HashMap*) и добавить его в *HashMap*, но уже с другим значением (вместо “CIRCLES” к примеру “SAME CIRCLES”).



Убедиться что размер *HashMap* не изменился, но значение по тому же объекту возвращается другое:



По желанию - почитать про HashMap.