Collections

equals() and hashCode()

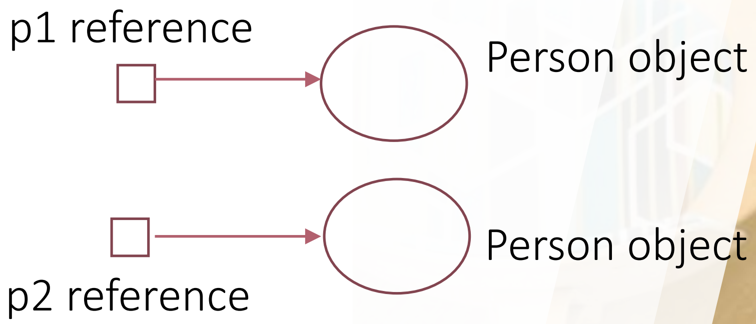
equals()

- Comparing two object references using the == operator evaluates to *true* only when both references refer to the same object i.e. == compares the bits in the references variables themselves and they are either equal or they are not.

- The *equals()* in *Object* behaves in the same way i.e. *equals()* in *Object* uses only the == operator for comparisons.

- Person p1 = new Person();

  Person p2 = new Person();

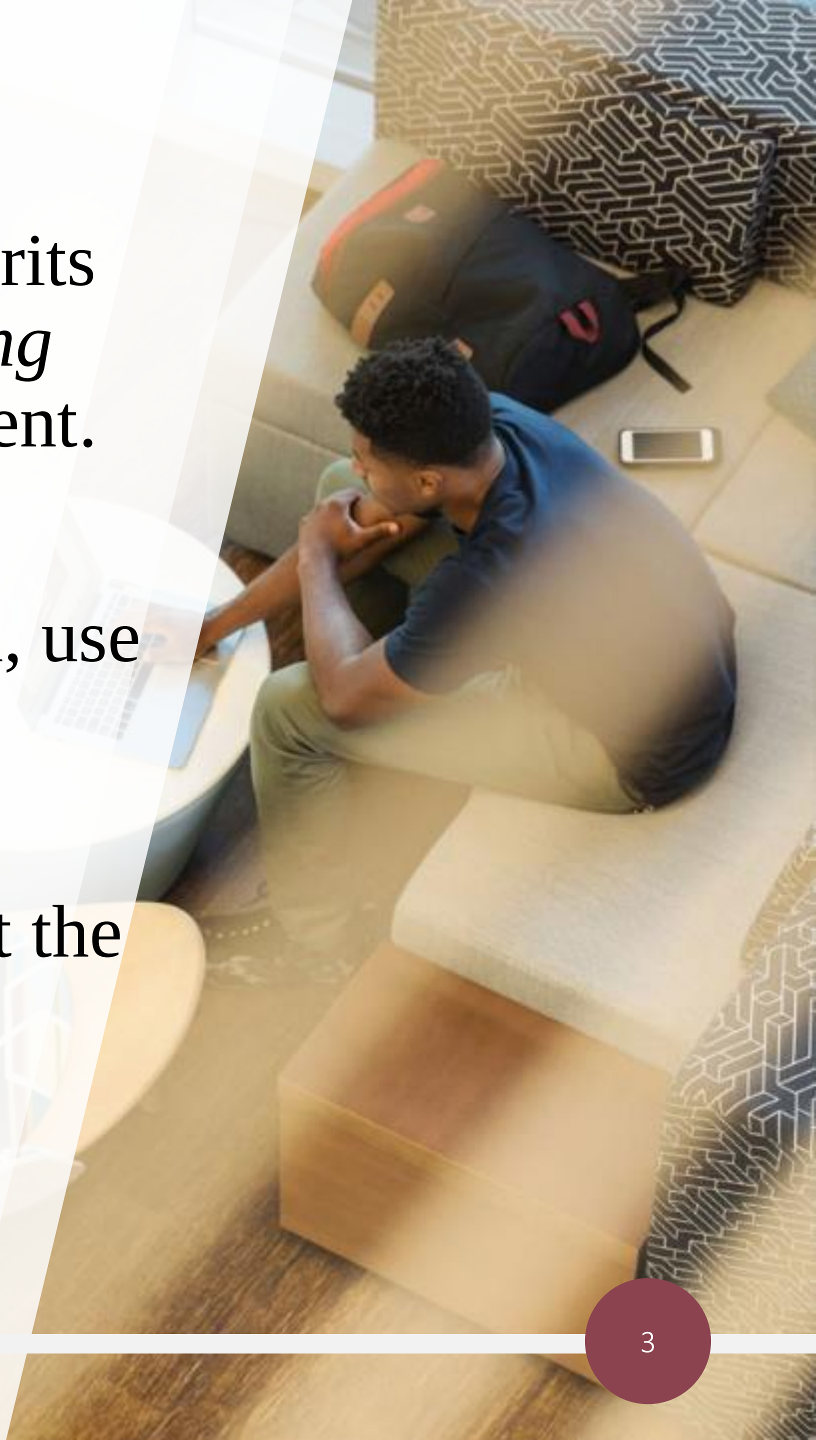


equals()

- The *String* class has overridden *equals()*, which it inherits from *Object*, so that you can compare two different *String* **objects** to see if their contents are meaningfully equivalent.

- When you need to know if two references are identical, use ==

- When you need to know if two objects themselves (not the references) are equal, use the *equals()* method.

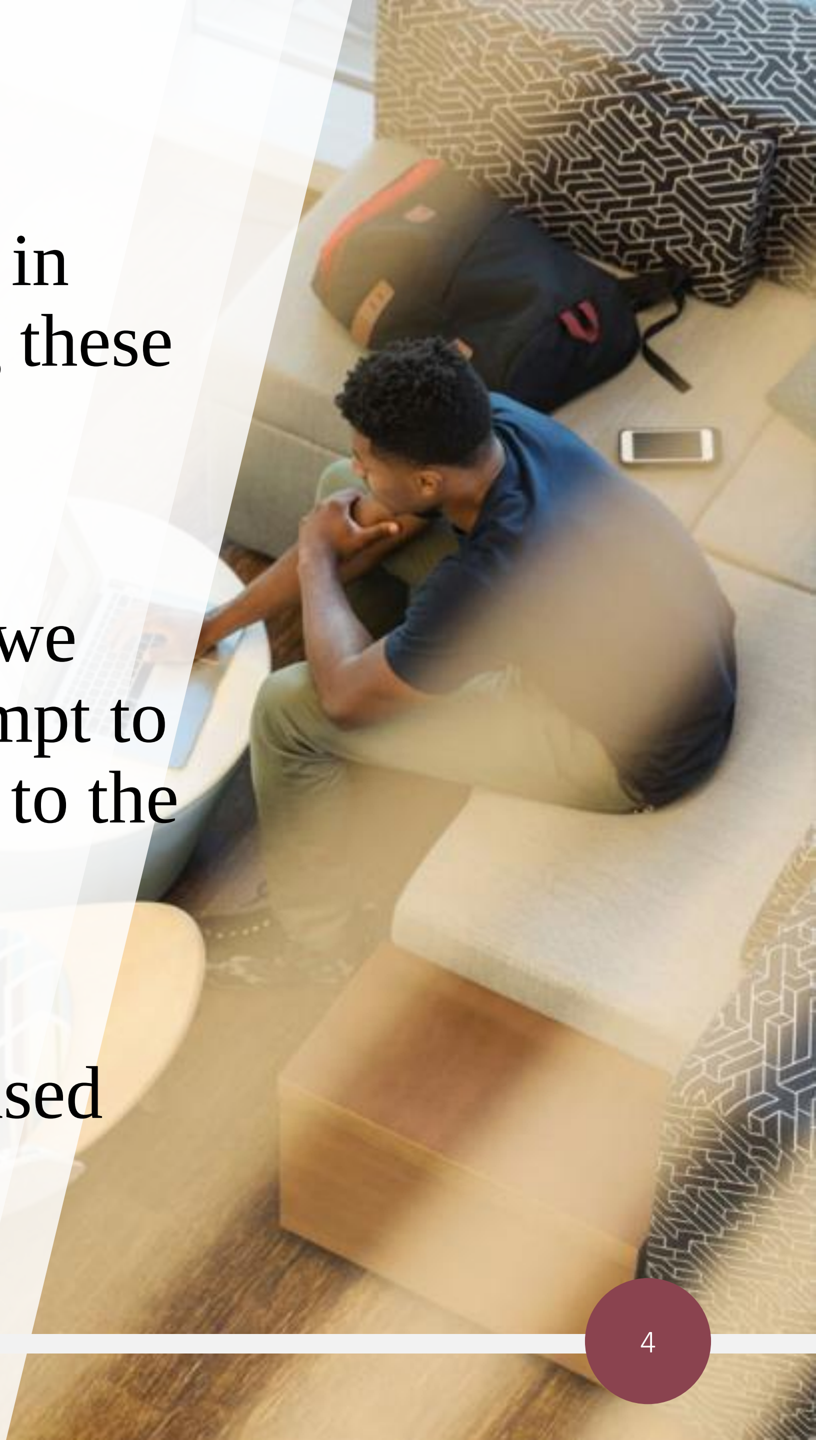


equals()

- Bear in mind what we will be storing objects (as keys) in Collections (such as *HashMap*) and searching/retrieving these objects again later.

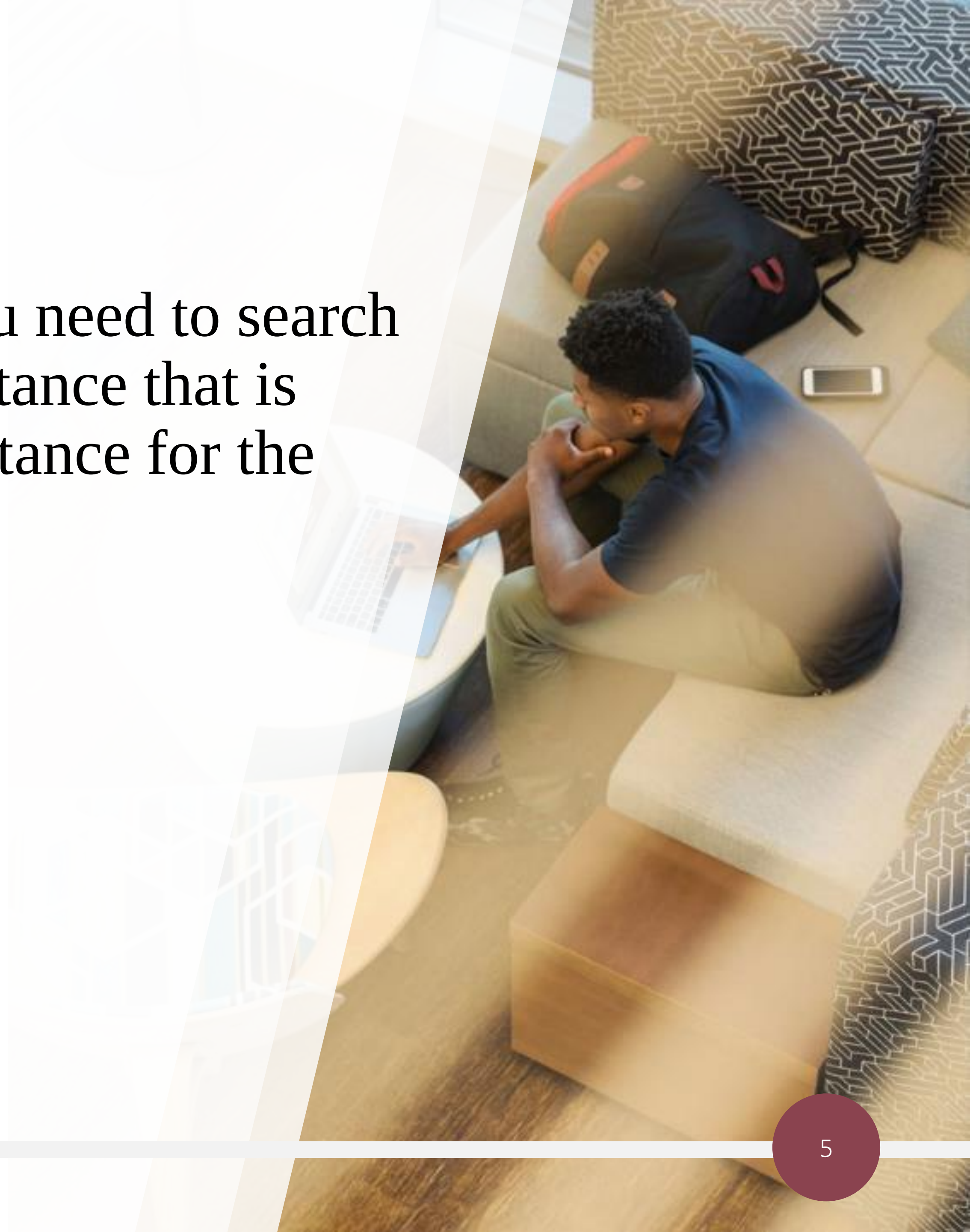
- For example, assume we do not override *equals()* and we store an object of that type in a Collection and later attempt to retrieve it - we are in trouble unless we have a reference to the exact object we used when storing the key.

- This is because the search for the key/object will be based on *Object::equals()* which, as we know, uses == on the references for equality testing.



equals()

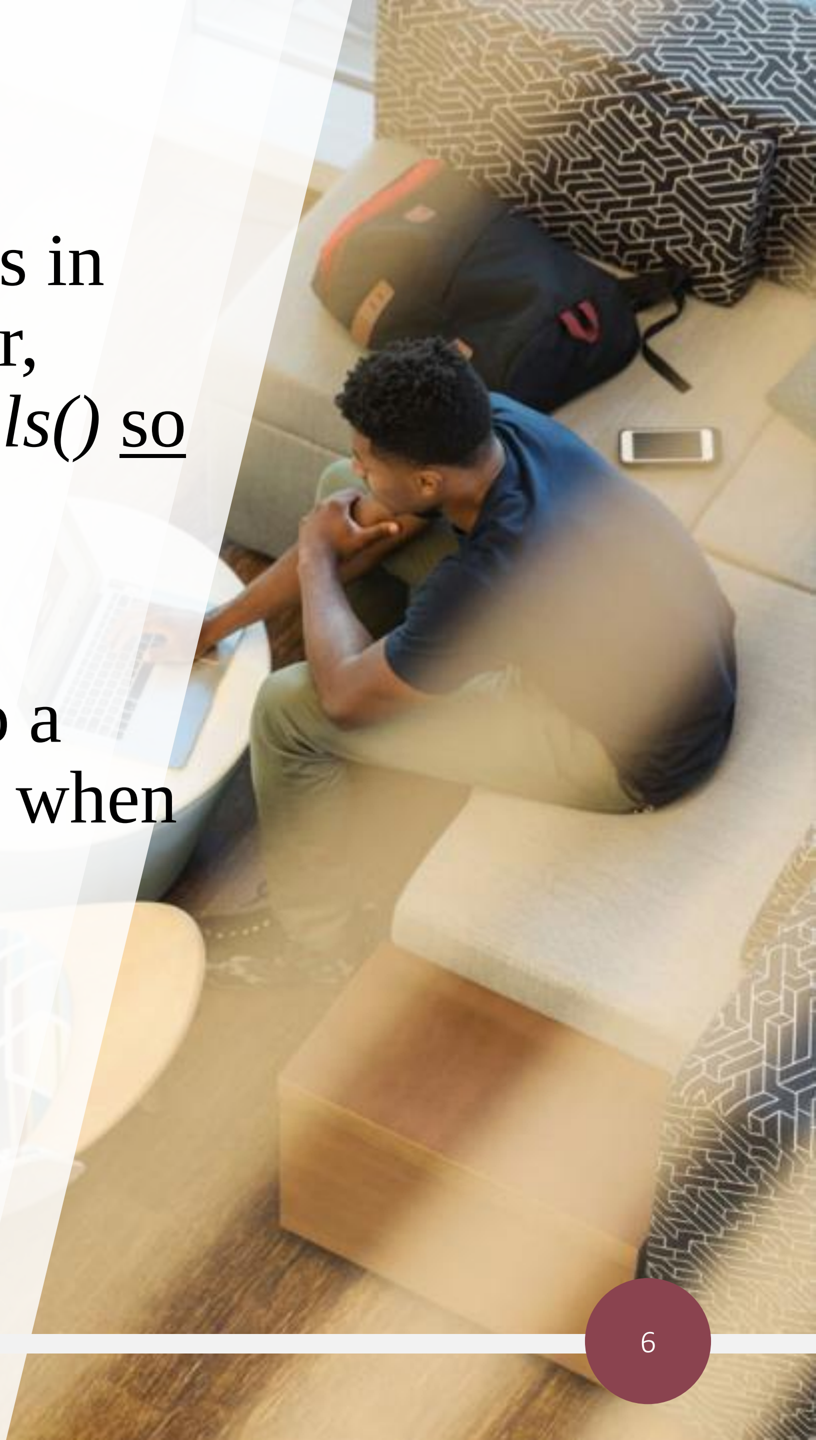
- Whereas, if you override *equals()*, when you need to search for object X, you can just re-create a *new* instance that is meaningfully equivalent to X and use that instance for the search.



equals()

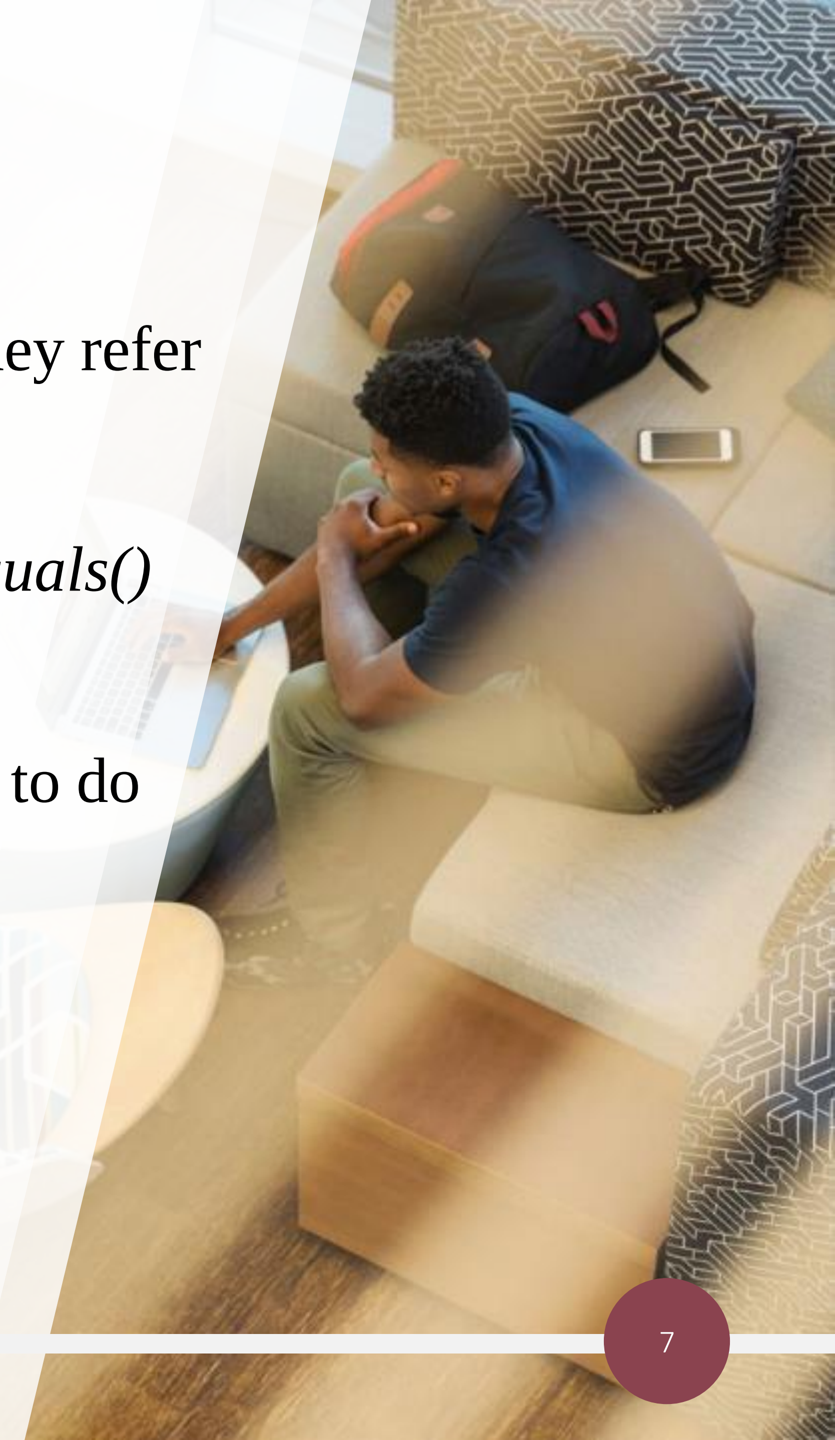
- If you want objects of your class to be used as elements in any data structure that uses equivalency for searching for, and/or retrieving an object, then you must override *equals()* <u>so that two different instances can be considered the same</u>.

- That way, you can use one instance when you **add** it to a Collection and essentially re-create an identical instance when you want to do a **search** based on that object as the key.

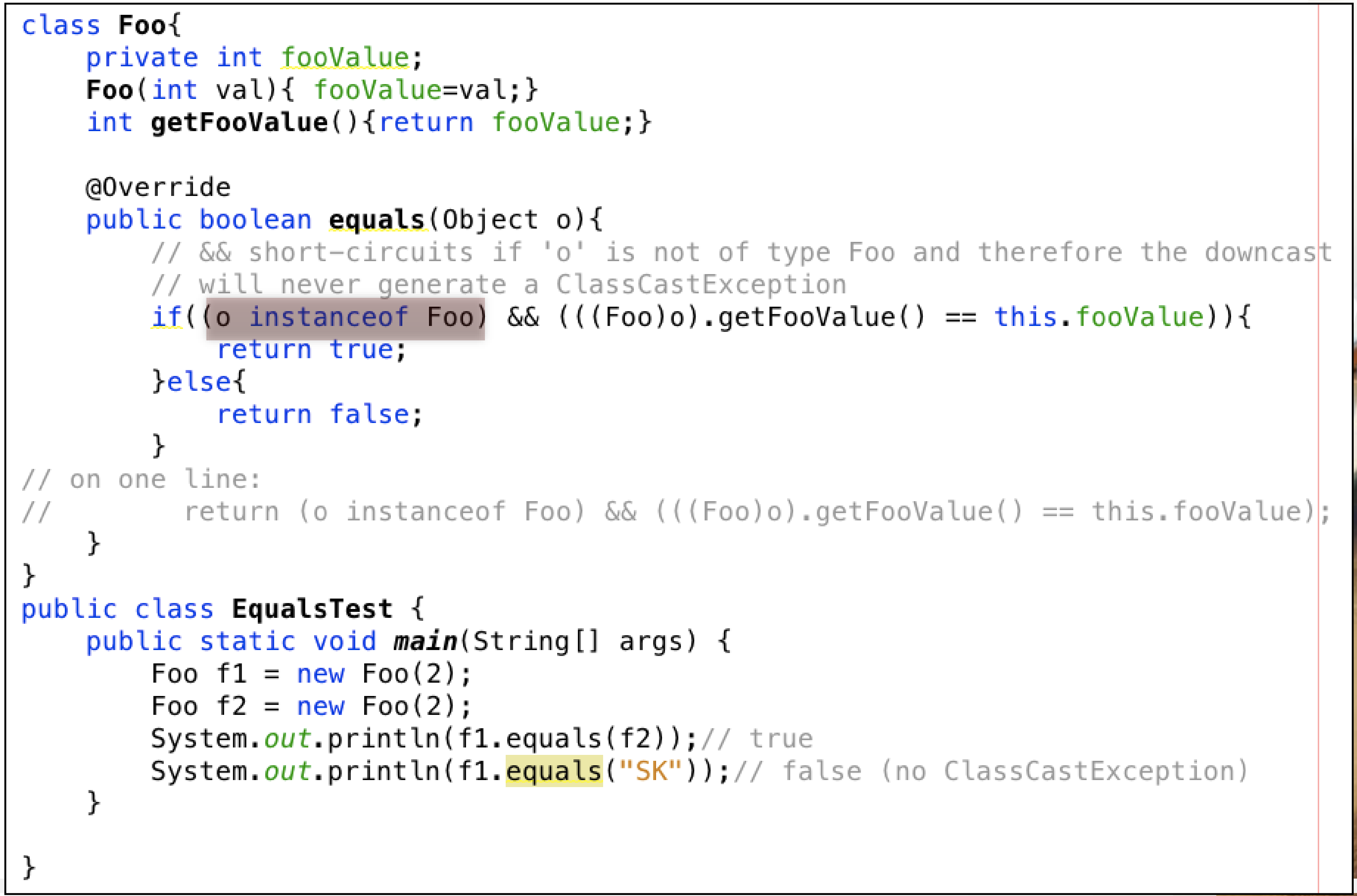


equals()

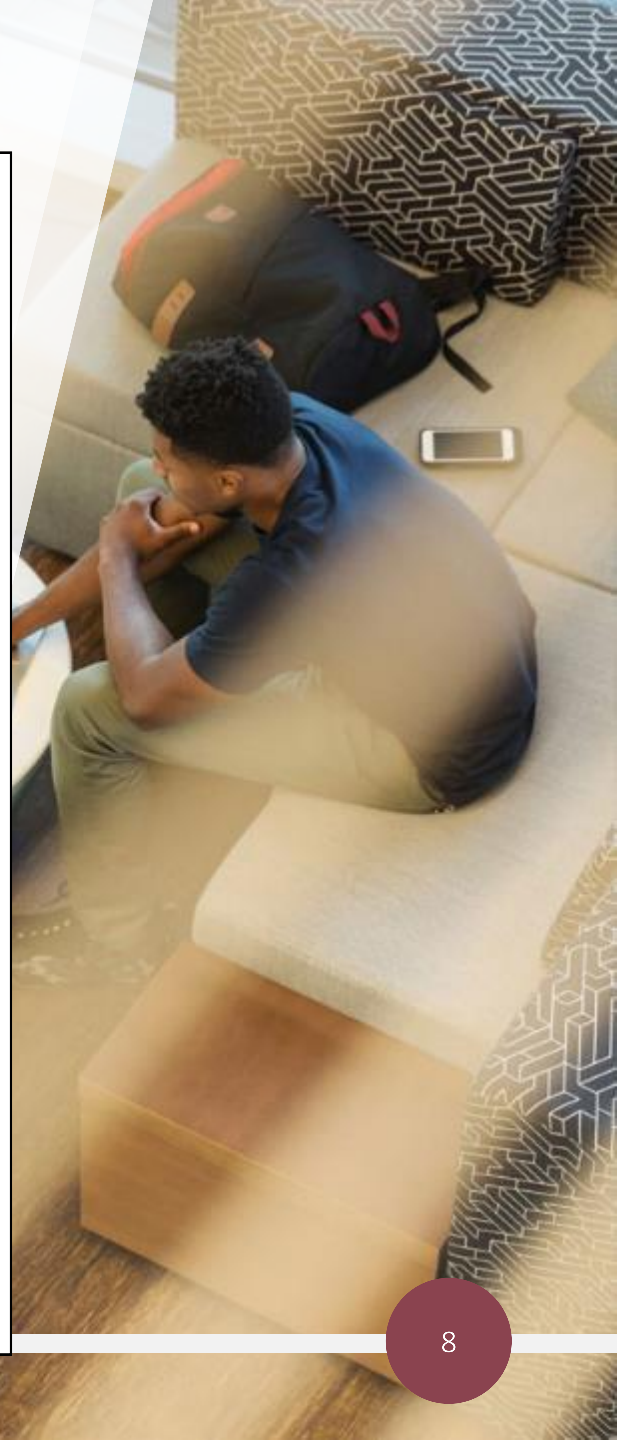
- In summary:
  - *Object:equals()* checks if two references are equal i.e. do they refer to the same object?
  - typically, this is not what you want so you must override *equals()*
    - *public boolean equals(Object obj)*
  - the *Object* passed in must be downcast to the relevant type; to do this safely use *instanceof*



equals()



```java
class Foo{
    private int fooValue;
    Foo(int val){ fooValue=val;}
    int getFooValue(){return fooValue;}

    @Override
    public boolean equals(Object o){
        // && short-circuits if 'o' is not of type Foo and therefore the downcast
        // will never generate a ClassCastException
        if((o instanceof Foo) && (((Foo)o).getFooValue() == this.fooValue)){
            return true;
        }else{
            return false;
        }
// on one line:
//          return (o instanceof Foo) && (((Foo)o).getFooValue() == this.fooValue);
    }
}
public class EqualsTest {
    public static void main(String[] args) {
        Foo f1 = new Foo(2);
        Foo f2 = new Foo(2);
        System.out.println(f1.equals(f2));// true
        System.out.println(f1.equals("SK"));// false (no ClassCastException)
    }

}
```



toString(), equals() and *hashCode()*

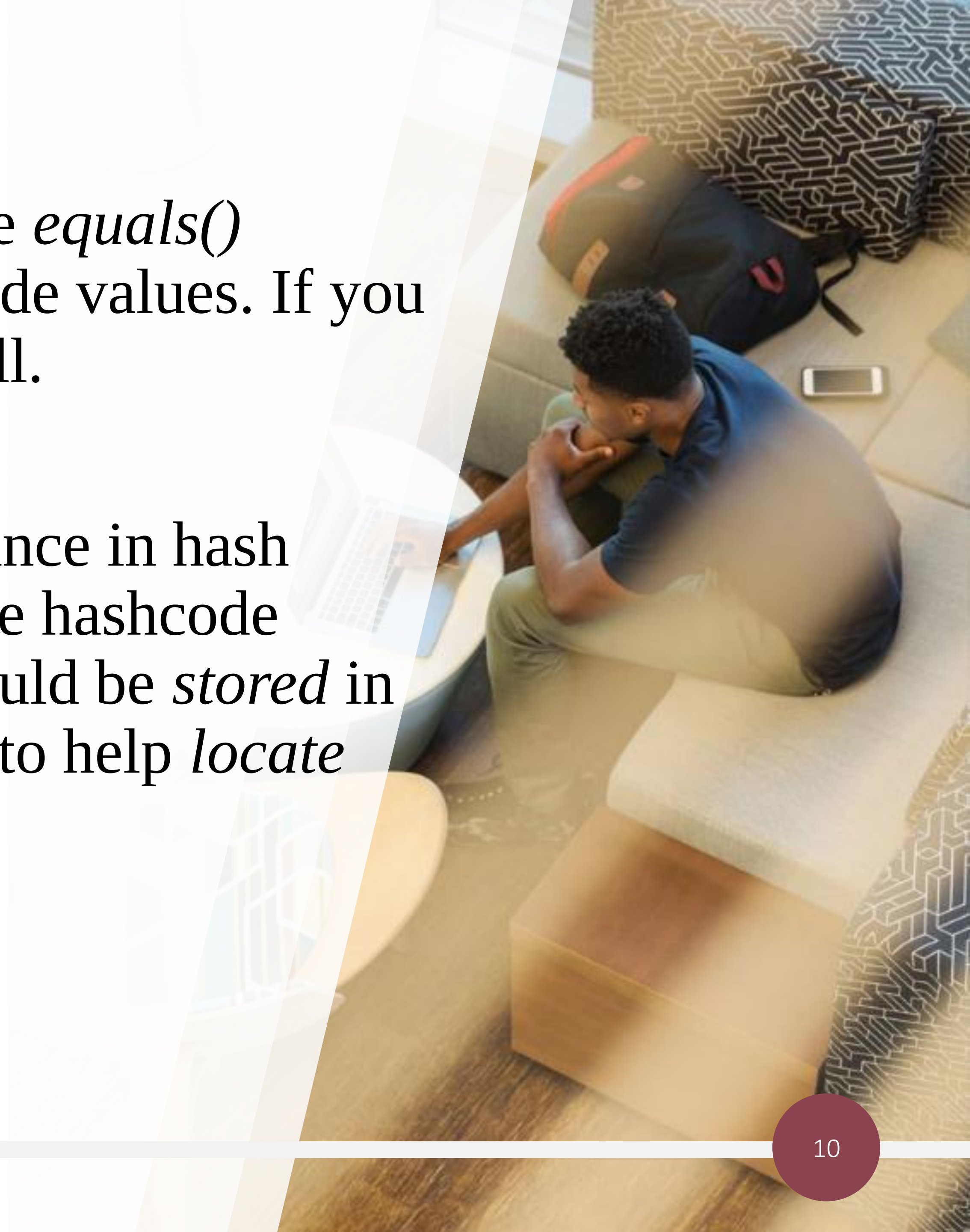
- Remember that *toString(), equals()* and *hashCode()* are all **public.** Therefore the following is an illegal override :
  - ➢ *class Foo{boolean equals(Object o){return true;}}* // should be <u>*public*</u>

- The following is also an illegal override :
  - ➢ *class Foo{public boolean equals(Foo f){return true;}}* // parameter should be *Object* not *Foo*

hashCode()

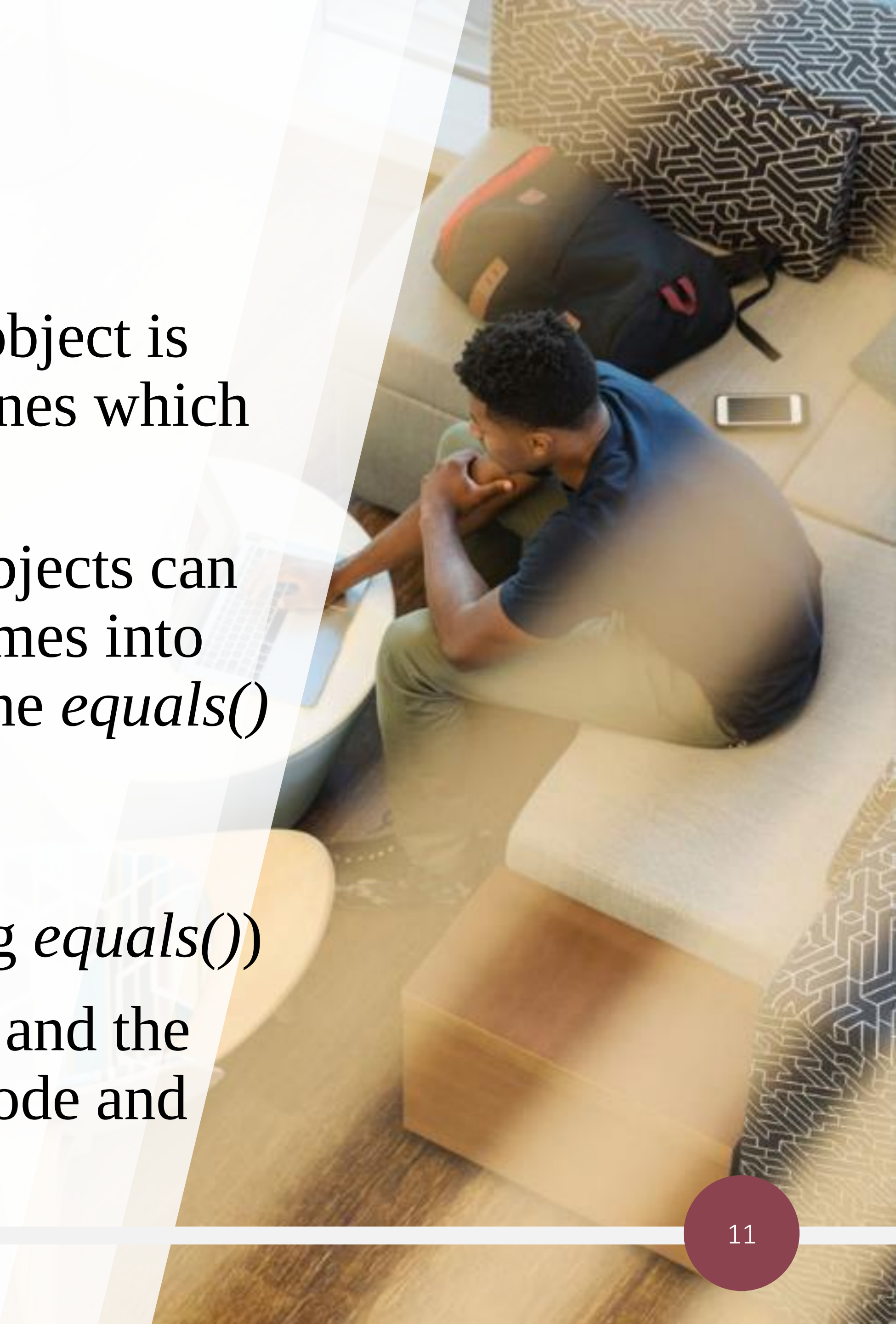
- If two objects are considered equal using the *equals()* method, then they must have identical hashcode values. If you override *equals()*, override *hashCode()* as well.

- Hashcodes are used for improving performance in hash based collections e.g. *HashSet*, *HashMap*. The hashcode value is used to determine how the object should be *stored* in the collection and the hashcode is used again to help *locate* the object in the collection.



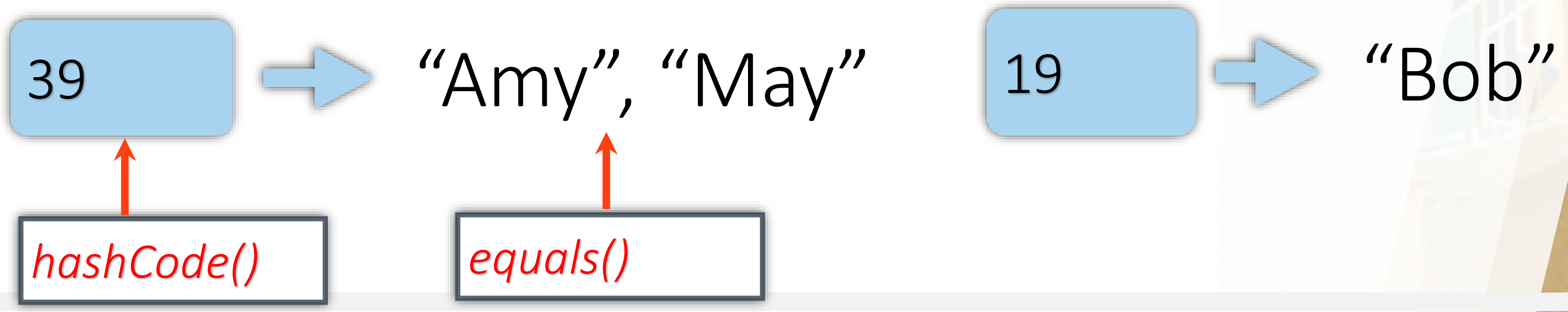
Understanding hashing

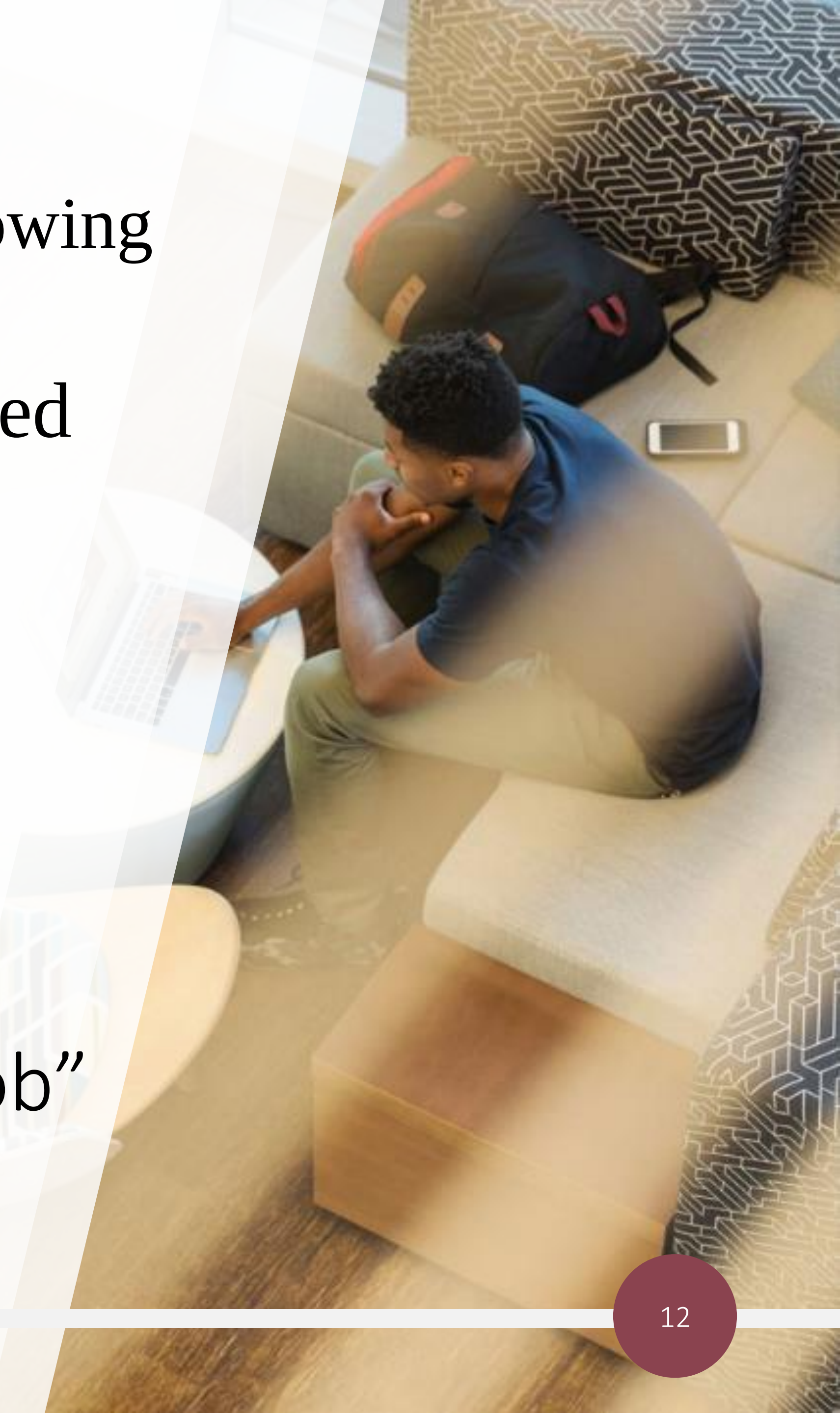
- Hashing is similar to putting items in buckets:
  - the hashcode value determines which bucket the object is stored in and later on, the hashcode value determines which bucket is searched to locate the object.
  - hashcodes are not necessarily unique so several objects can land in the same bucket; this is where *equals()* comes into play - the correct object is then located by using the *equals()* method
    1. find the right bucket (with *hashCode()*)
    2. search the bucket for the right element (using *equals()*)
  - thus, for an object to be located, the search object and the object in the collection must have the same hashcode and return *true* for *equals()*.



Understanding hashing

- Assume we are storing names according to the following hashing calculation algorithm - A=1, B=2 etc…
- The numbers associated with each letter are added together to give the hashcode (bucket number)
  - Bob = B(2) + O(15) + B(2) = 19
  - Amy = A(1) + M(13) + Y(25) = 39
  - May = M(13) + A(1) + Y(25) = 39





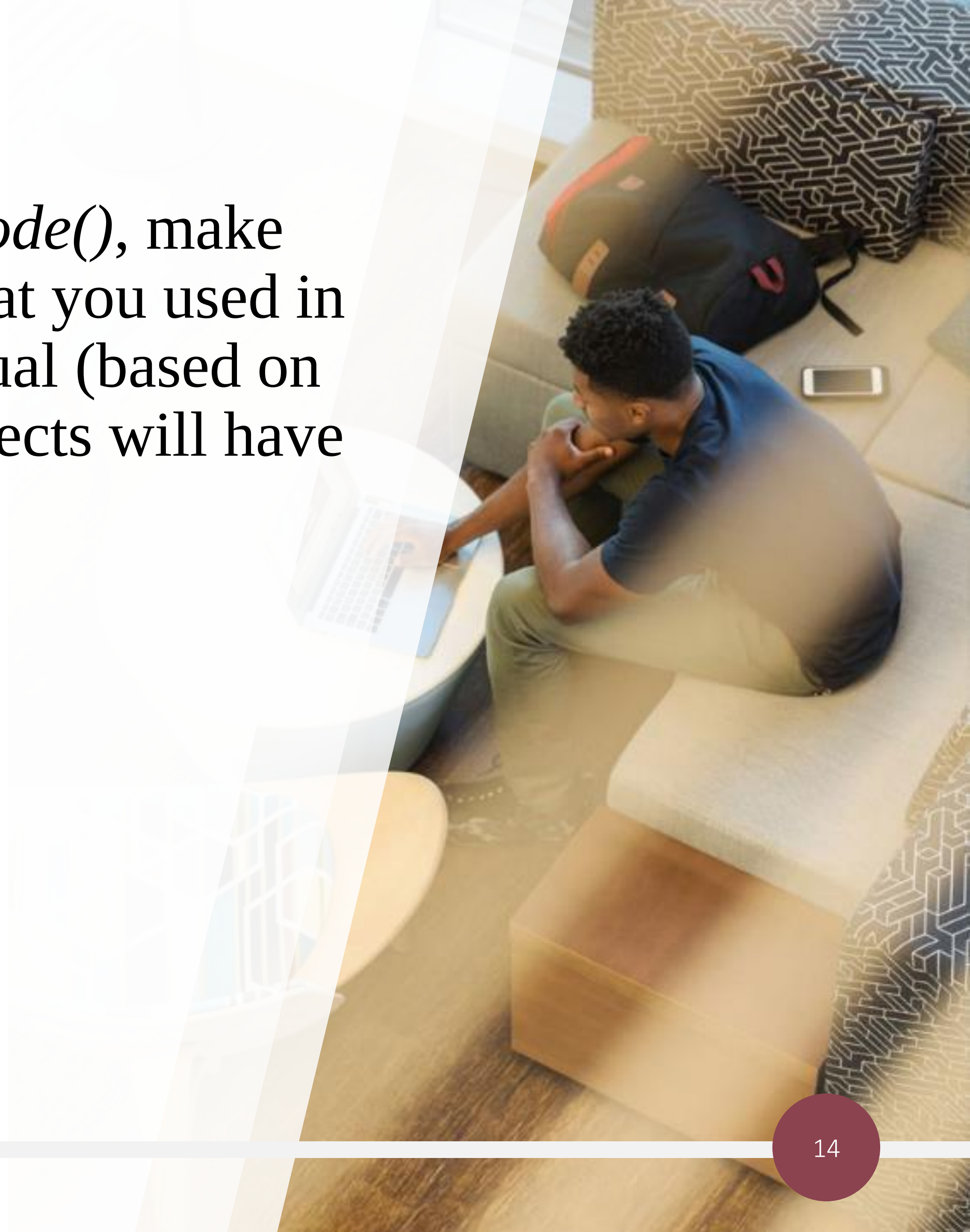
hashCode()

- The default *hashCode()* method in *Object* always comes up with a unique number for each object, even if the *equals()* method is overridden and states that two or more objects are equal.

- In other words, it does not matter how equal they are if their hash codes do not reflect that (as you will be directed to the wrong bucket). Therefore the *hashCode()* contract states that, **if two objects are equal, their hashcodes must be equal as well**.

hashCode()

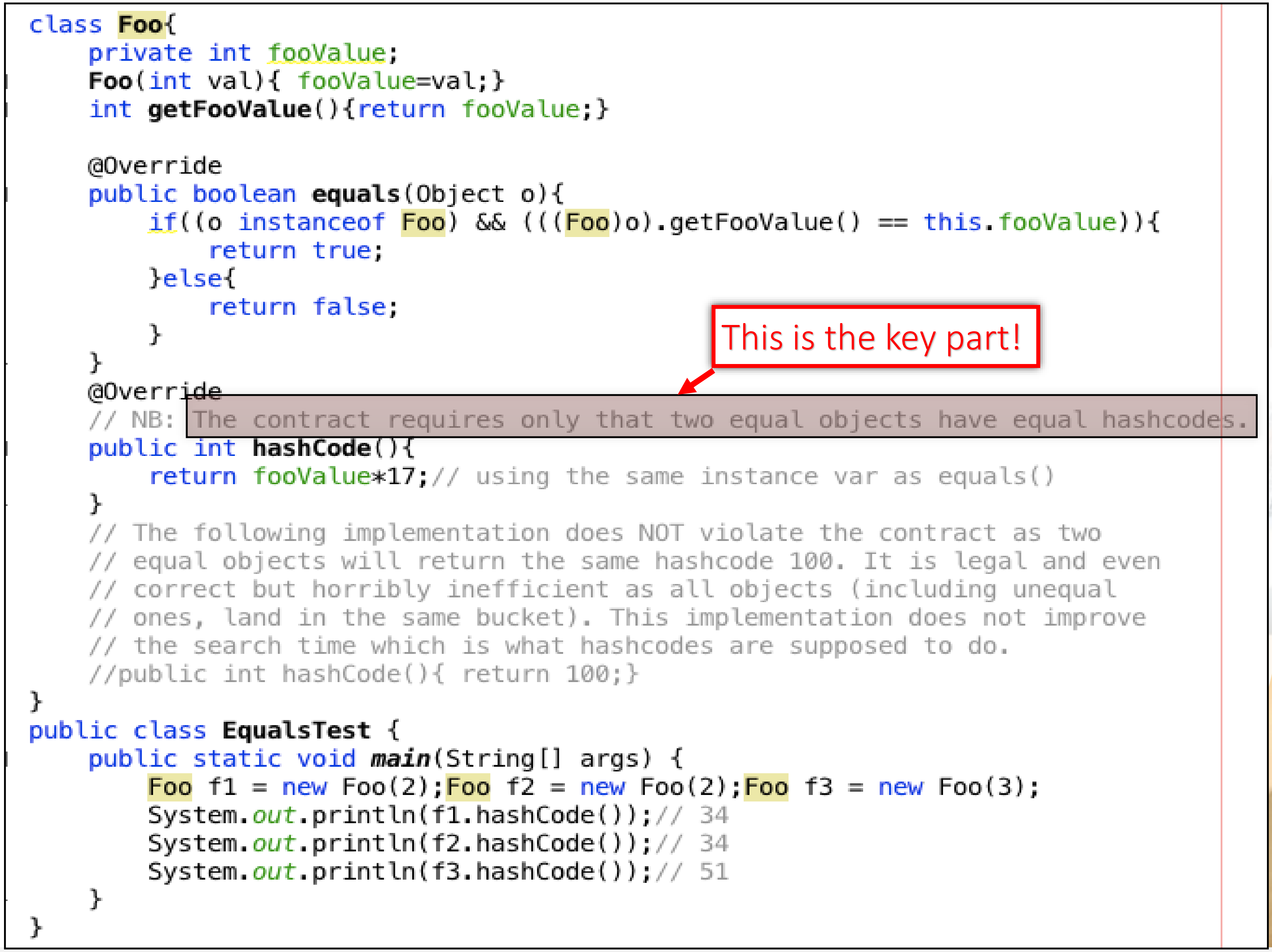
- When calculating the hashcode in *hashCode()*, make sure to use the same instance variables that you used in *equals()*. Therefore, if two objects are equal (based on their instance variables), then the two objects will have the same hashcode value.



hashCode()

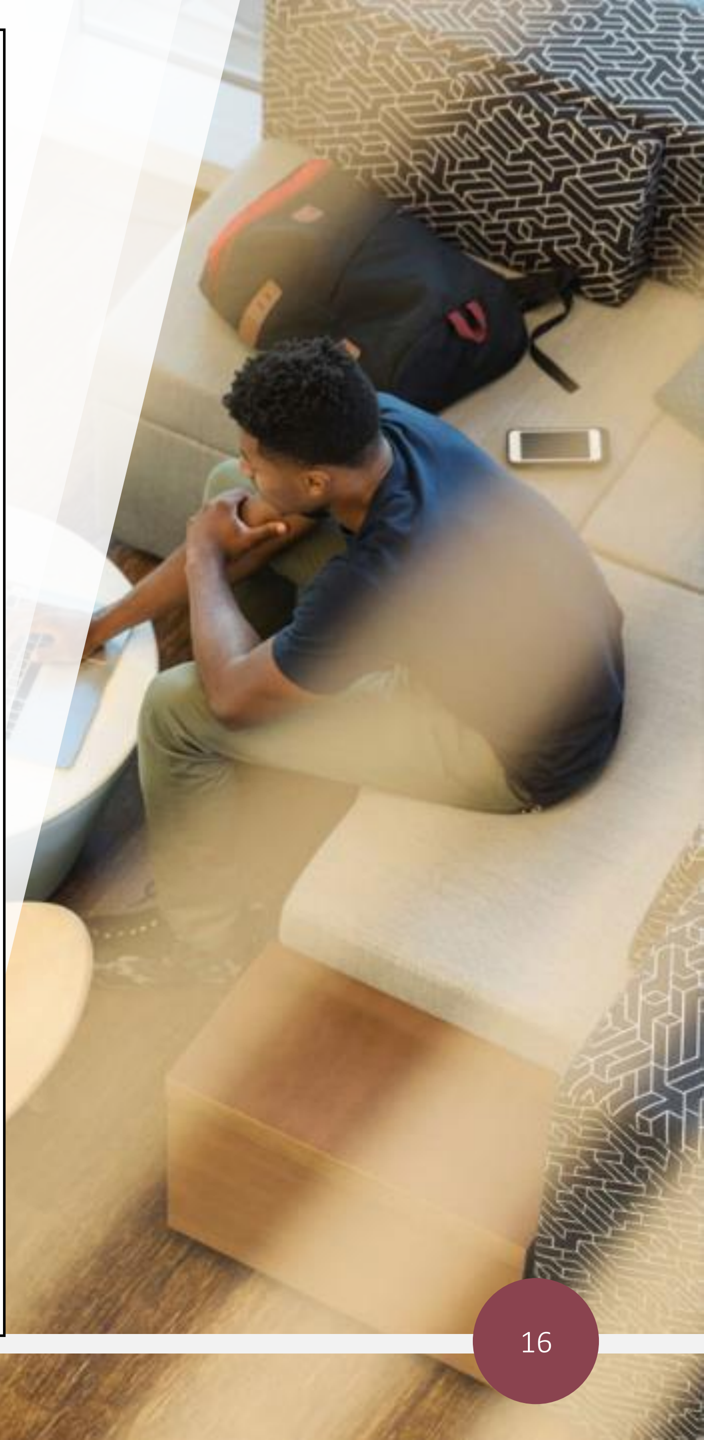
- Note: do not use *transient* instance variables in the hashcode calculation as these are not serialised.

- For example, if a *transient* variable has 10 as its value at the time you *store* the object in a *HashMap*, then you serialise the object to disk (*transient* not serialised); you then deserialise the object, the *transient* variable will get a default value e.g. 0 and therefore the hashcode value will be different when you go to *locate* that object (i.e. wrong bucket).



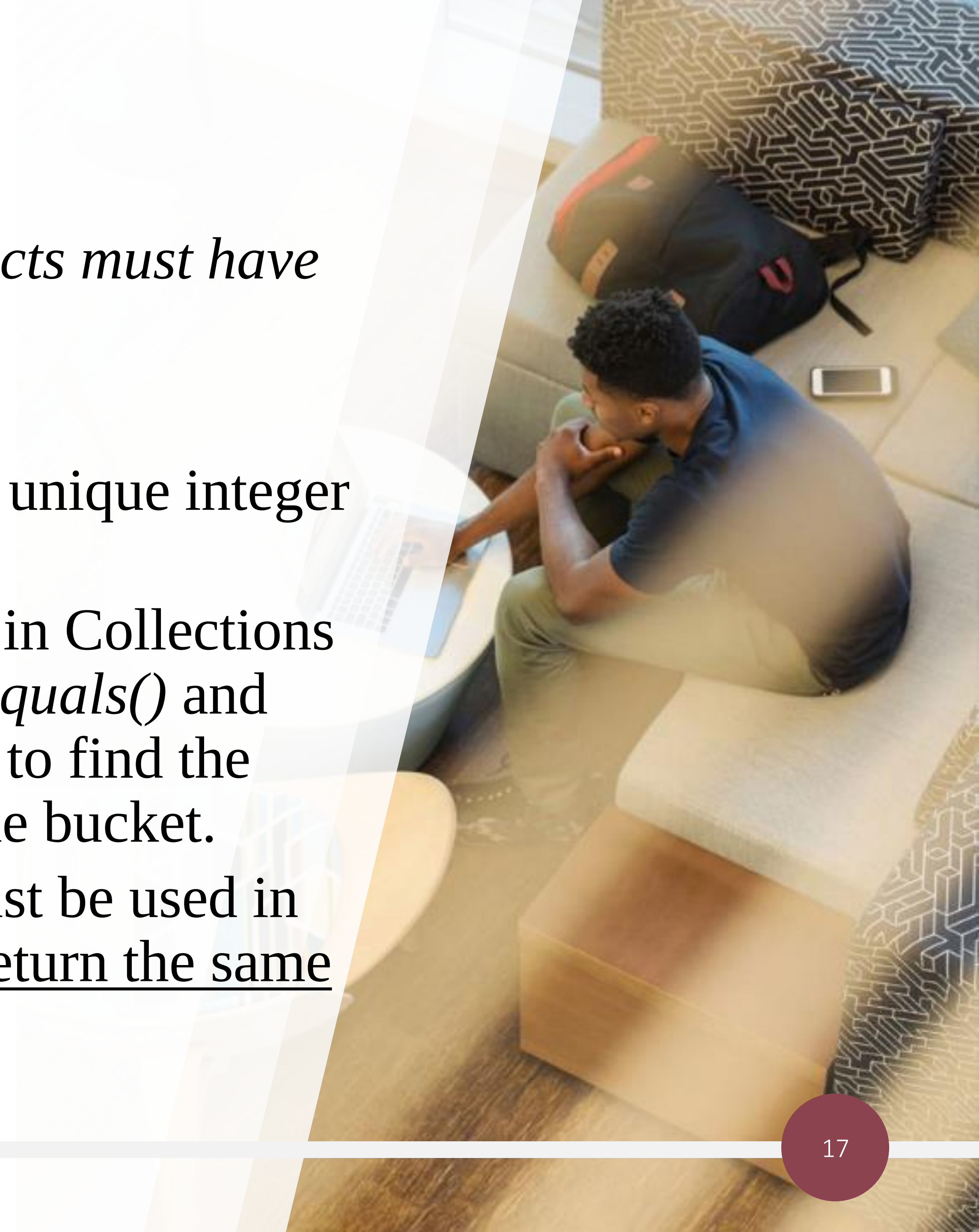
```java
class Foo{
    private int fooValue;
    Foo(int val){ fooValue=val;}
    int getFooValue(){return fooValue;}

    @Override
    public boolean equals(Object o){
        if((o instanceof Foo) && (((Foo)o).getFooValue() == this.fooValue)){
            return true;
        }else{
            return false;
        }
    }
    @Override
    // NB: The contract requires only that two equal objects have equal hashcodes.
    public int hashCode(){
        return fooValue*17;// using the same instance var as equals()
    }
    // The following implementation does NOT violate the contract as two
    // equal objects will return the same hashcode 100. It is legal and even
    // correct but horribly inefficient as all objects (including unequal
    // ones, land in the same bucket). This implementation does not improve
    // the search time which is what hashcodes are supposed to do.
    //public int hashCode(){ return 100;}
}
public class EqualsTest {
    public static void main(String[] args) {
        Foo f1 = new Foo(2);Foo f2 = new Foo(2);Foo f3 = new Foo(3);
        System.out.println(f1.hashCode());// 34
        System.out.println(f2.hashCode());// 34
        System.out.println(f3.hashCode());// 51
    }
}
```

This is the key part!



hashCode() summary

- *hashCode()* contract:
  - the “contract” states that “*two equal objects must have the same hashcode*”.
- *public int hashCode()* :
  - by default, *hashCode()* in *Object* returns a unique integer for objects.
  - to be certain that your objects can be used in Collections that use hashing, you must override both *equals()* and *hashCode()* as both are used - *hashCode()* to find the bucket and *equals()* to find the object in the bucket.
  - the instance variables used in *equals()* must be used in *hashCode();* that way, <u>equal objects will return the same hashcode integer value.</u>



ContactTest program

MutableFieldsTest program