Working with Objects

Paolo Perrotta

Developer, Author

@nusco | www.paoloperrotta.com

This is a module about creating and using objects.

Creating an Object

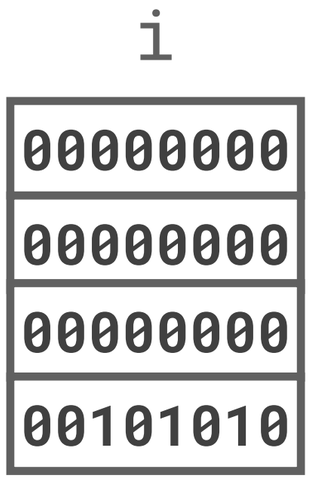
```
Alarm alarm = new Alarm("Temperature too high");
```

Using an Object

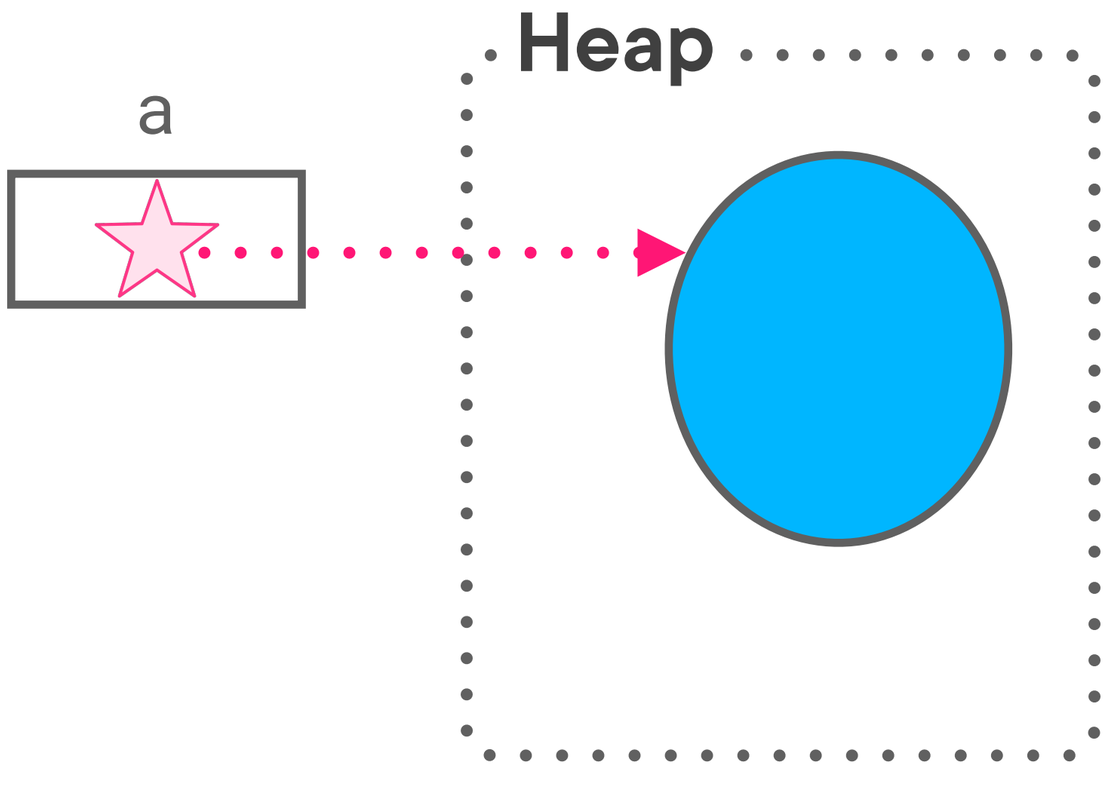
```
Alarm alarm = new Alarm("Temperature too high");
alarm.active              → false
alarm.message             → "Temperature too high"
alarm.getReport();        → ""
alarm.turnOn();
alarm.active              → true
alarm.getReport();        → "Temperature too high"
```

Primitives vs. Objects

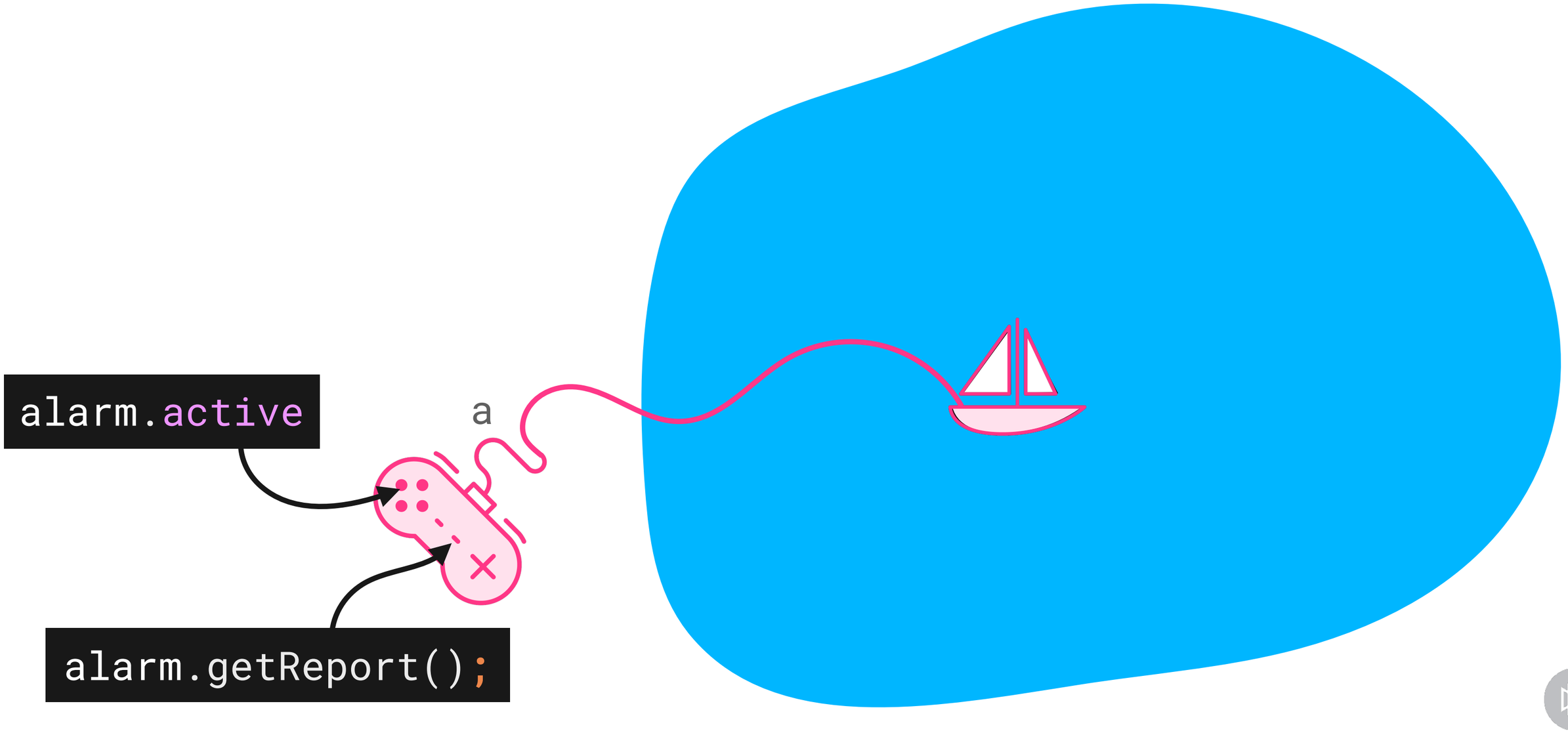
```
int i = 42;
```




```
Alarm a = new Alarm("");
```

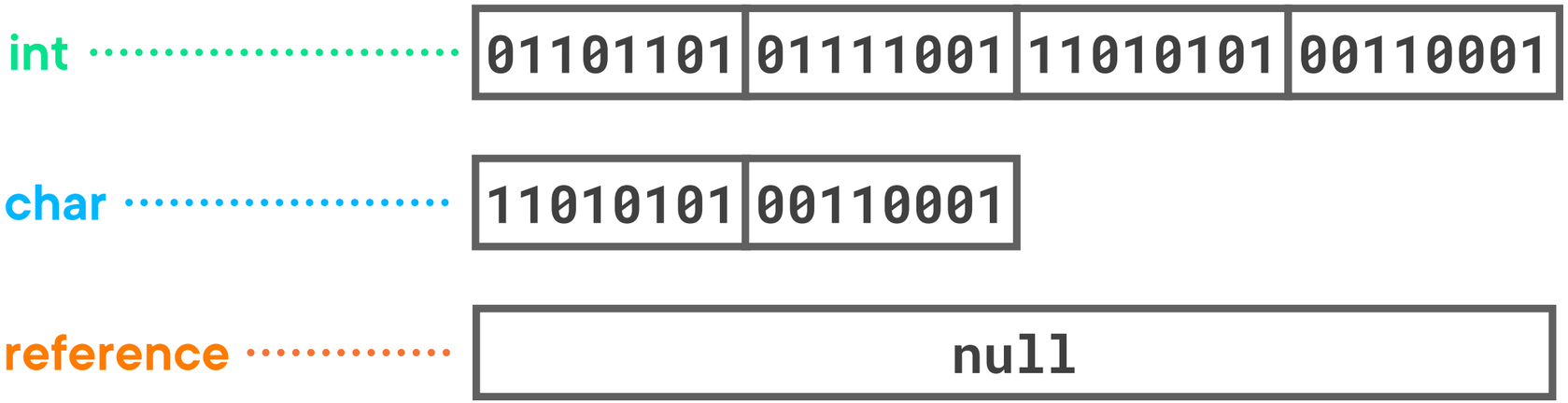


Object References

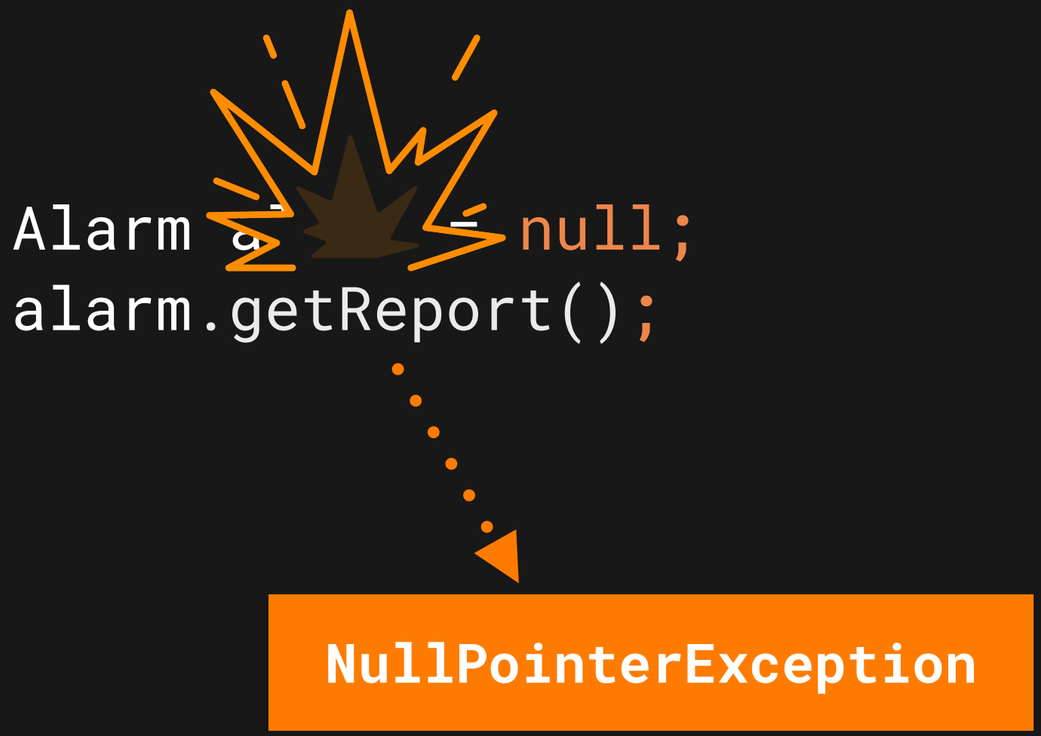


A reference might not have an object.

Primitive Values and Reference Values



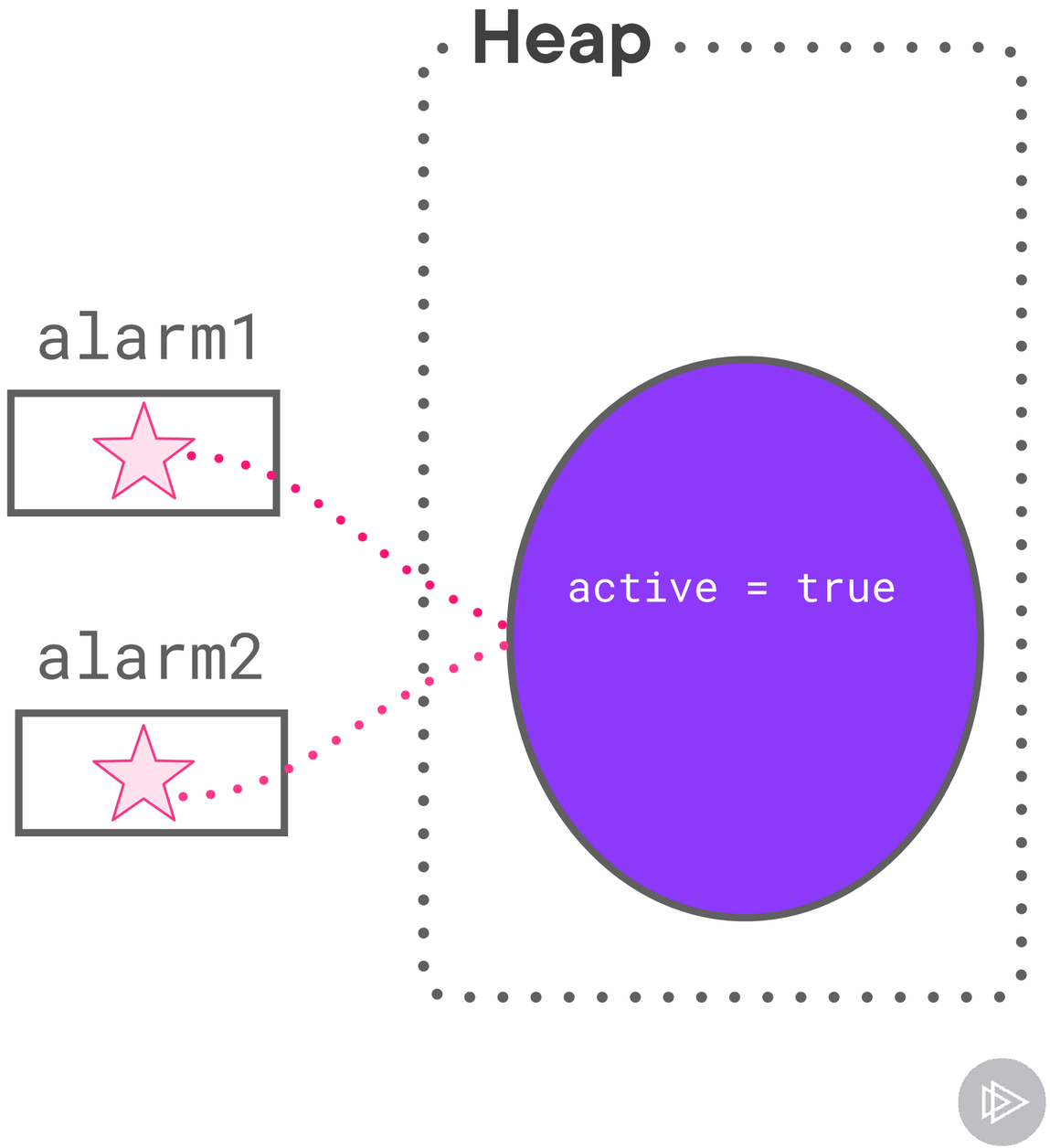
Null References



It’s hard to avoid null pointer exceptions completely.

Aliasing

```
Alarm alarm1 = new Alarm("");
alarm1.active    ⟶ false
Alarm alarm2 = alarm1;
alarm2.turnOn();
alarm2.active    ⟶ true
alarm1.active    ⟶ true
```



Primitives have no Aliasing

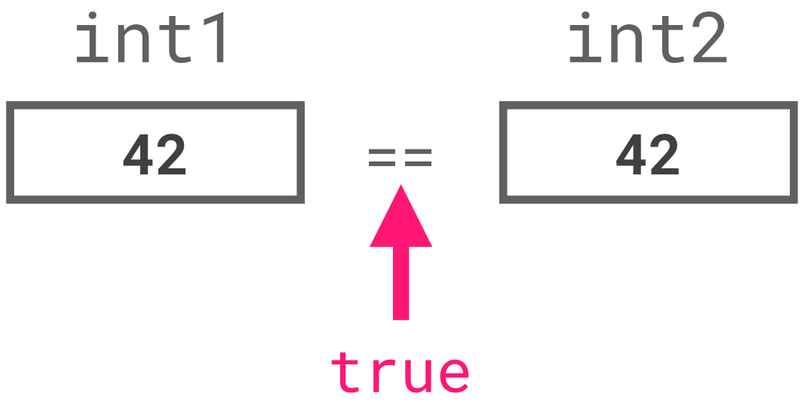
```
int integer1 = 41;
int integer2 = integer1;
integer1++;
integer1      → 42
integer2      → 41
```

Primitives are passed by value. Objects are passed by reference.

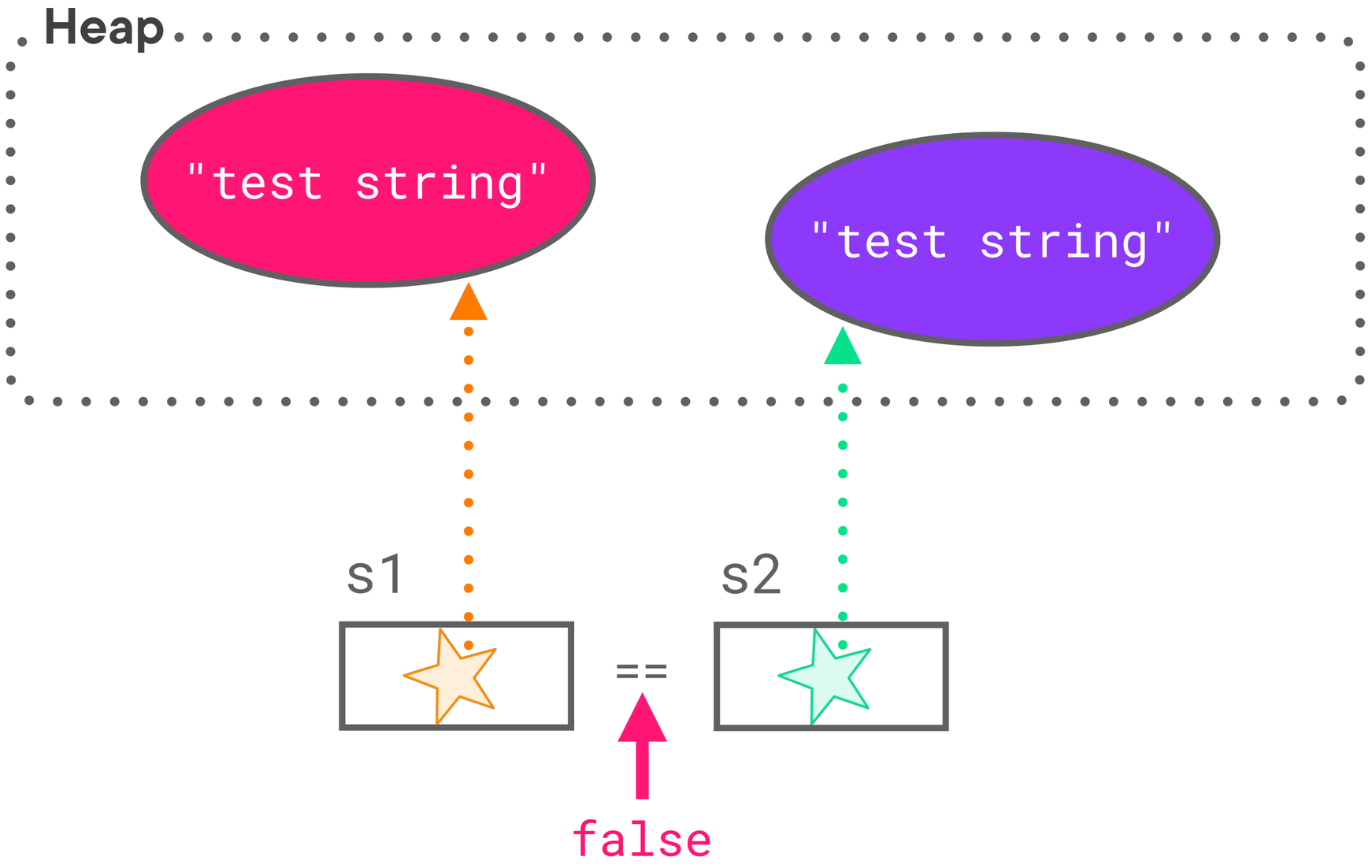
A Painful Memory

```
String s1 = "test string";
String s2 = "test string";
s1 == s2    ⟶ false
```

Comparing Primitives



Comparing Strings



In some classes, *equals()* compares for identity. In others, it compares for equality.

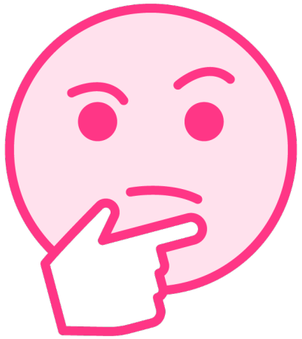
Comparing Objects with *equals()*

```
String s1 = "test string";
String s2 = "test string";
s1.equals(s2)    ⟶ true
```

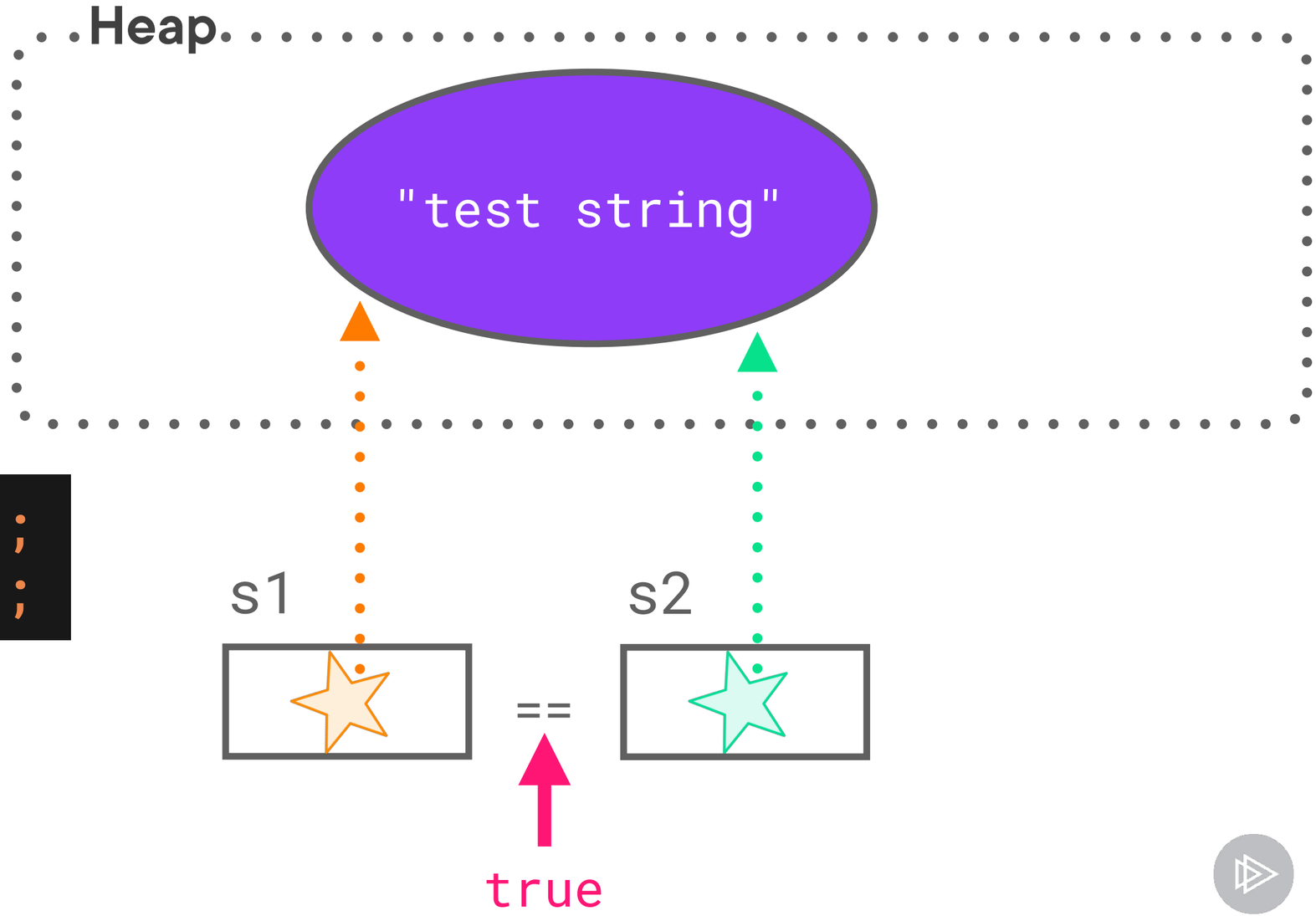
My Broken Code

```
String s1 = "test string";
String s2 = "test string";
s1 == s2    ⟶ false
```

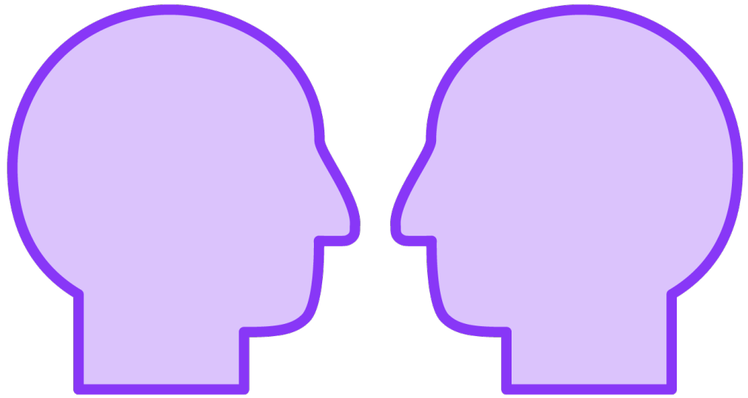
String Interning



```
String s1 = "test string";
String s2 = "test string";
```



Identity and Equality



Two objects are...

- ...*identical* if they are the same object in memory
- ...*equal* if they contain equal data

The == operator checks for identity

- Strings are a confusing special case

The *equals()* method checks for...

- ...well, it depends on the class.

Primitives have only equality, no identity

Final Variables

```
final int i = 42;
i = 10;
i++;
```

Final Object Variables

```
final Alarm alarm1 = new Alarm("");
alarm1 = new Alarm(“Another alarm”);
alarm1 = null;
```

Final Object Variables

```
final Alarm alarm1 = new Alarm("");
alarm1.active   ⟶ false
alarm1.turnOn();
alarm1.active   ⟶ true
```

“Constant” vs. “Immutable”

Constant

Applies to variables, including object references. It means that you cannot reassign the variable.

Immutable

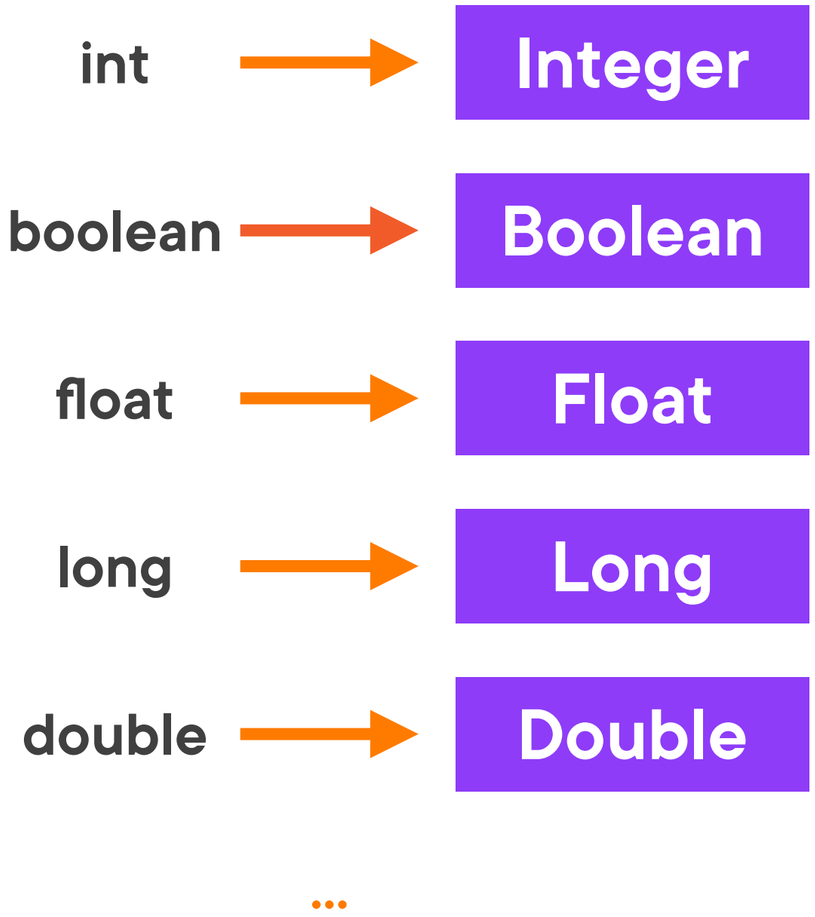
Applies to objects (for example, strings). It means that you cannot change the state of the object.

Immutability is a design concept.

Putting Variables in a Collection

```
int aPrimitive = 42;
Integer anObject = new Integer(aPrimitive);
myList.add(anObject);
```

Wrapper Classes



Autoboxing

```
int aPrimitive = 42;
myList.add(aPrimitive);
```

Autoboxing

```
int aPrimitive = 42;
Integer anObject = aPrimitive;
```

Autounboxing

```
int aPrimitive = 42;
Integer anObject = new Integer(aPrimitive);
int anotherPrimitive = anObject;
```

Summary

Object variables contain references

- References can be null
- You can have aliased references
- Object identity is not the same as equality
- References can be constant, not objects

Java can autobox/unbox primitives for you

Up Next:

Defining Your Own Classes