Sunday, December 30, 2018 1:53 PM

Primitives vs Objects

Microsoft: DEV277x
Object Oriented Programming in Java

Primitives vs Objects
In Java there are two general types of data: primitives and Objects

Primitives

There are 8 different types of primitive data
primitiveexampledescription
intint a = 1;32-bit signed whole number
doubledouble b = 1.0;64-bit real number
booleanboolean c = true;logical data type can only be true or false
charchar d = 'x';stores a single unicode character
bytebyte e = 2;8-bit signed whole number
shortshort f = 3;16-bit signed whole number
longlong g = 3000000000;64-bit signed whole number
floatfloat h = 2.0;32-bit real number
Aspects of primitive data:

- when working with primitive data types Java stores them directly in memory
- when you copy a primitive variable, only the data is copied, creating a separate variable that holds the same value
- when working with primitives you can use mathematical operators to perform direct calculations like +, -, / and *
- you can use shortcuts to update the value of a primitive value like ++, -- or +=

Objects

Aspects of Object data:

- an object is a method of storing multiple pieces of data and the methods who act on this data under a single data type
- this means within a single object you could have multiple pieces of data including both primitives and other objects

- because objects have the potential to store large amounts of data Java stores them in memory differently from primitives, storing the location for the data in the actual variable as a "reference"
- when you are working with Objects you can't use the mathematical operators or shortcuts, instead you need to rely upon the methods provided by the object like "string".equals("string") or scanner.nextInt()

Null

Each primitive has its own associated "zero value". This is a value that java uses as a stand in when creating a place in memory for a primitive before you actually give it a value to store.
All object types share a single "zero value"- null
Null means literally "no object", which is different from an empty object. For example, if you try to use the String's lengh method on a null string you get an error. However, calling length on an empty string is perfectly valid.

```
String temp; // if you don't initialize a String it defaults to null
temp.length(); // causes a "Null Pointer Exception"
String empty = "";
empty.length(); // returns 0;
```