Java Fundamentals 2

Arrays

Arrays

What follows is a brief slide running over a quick recap from Arrays in PHP, and an overview of the Java's array functionality – don't worry if a lot of it looks confusing in the overview, we'll be covering it in greater detail later on in these slides.

Arrays – a quick recap

- Arrays are a fundamental data structure to almost every programming language. So far, in PHP, you've seen;

- Indexed array;
  Set Structure: $Array[i] = x (where x is the datatype in use – i.e: String)

  Get Structure: $Array[i] – where i is an integer, pointing to the element you wish to retrieve

- Associative array;
  Set Structure: $Array["String"] = x

  Get Structure: $Array["String"]

  Note that in associative Arrays, we access our data via a String – rather than just an integer.

- While php calls these "Arrays", they are actually Hash maps.
- In Java, we're going to use real Arrays.

Arrays in Java - Overview

- An array is in short, a collection of variables, all of the same type.
- An array is of FIXED length;
  - You MUST tell Java how many variables will be held in an array, BEFORE adding them!
- Arrays are accessed by the usual Key => Element approach.
  (We'll go over this in detail shortly)
- Arrays start at position 0 (Zero) – trying to access a position below 0, or above your max Array size will result in an error (typically at run-time).
- Arrays have their own selection of methods, to help you interact with them.

Arrays in Java

- As mentioned previously, when setting up our Array, there are a few simple rules;
  - Arrays can only hold ONE type of data – i.e: ONLY integers, or ONLY Strings, etc.
  - We need to declare how big our Array will be, BEFORE we put data into it.

- So let's build an array – we want 5 Strings, all holding the name of a Beer.

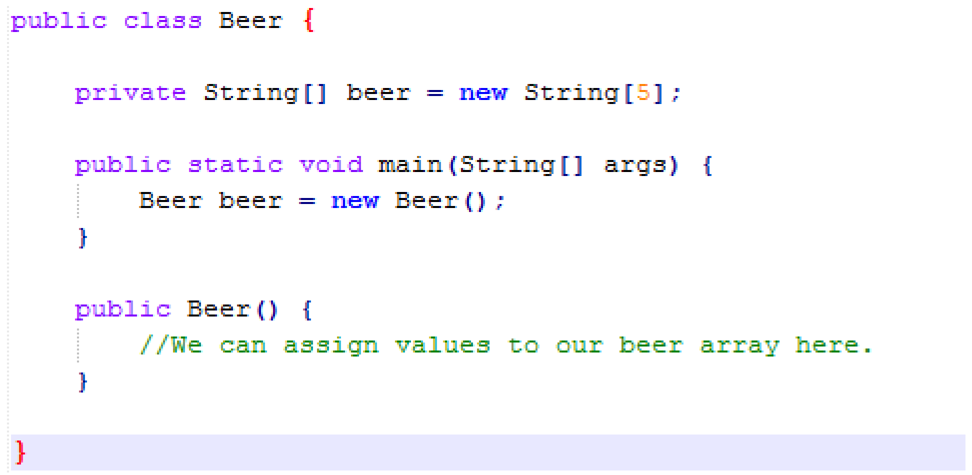
- The structure for building an array is as follows;

  Type *any-name-you-want* = new Type[*size-of-array*];
  - Where Type is the data type (i.e; String, int, double)
  - *size-of-array* is how many variables our array will hold (integer)
  - *any-name-you-want* is the name of the array.

- Our structure for building an array for holding 5 Strings, containing Beer names;

```
String beer = new String[5];
```

Arrays in Java



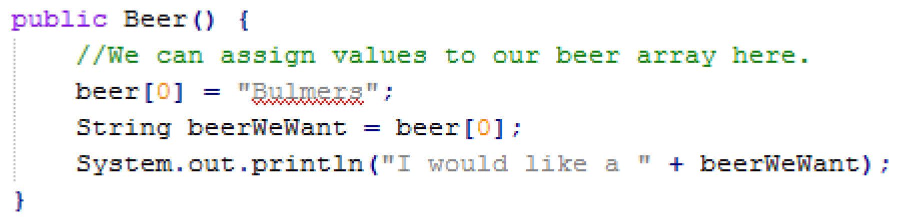
```
public class Beer {

    private String[] beer = new String[5];

    public static void main(String[] args) {
        Beer beer = new Beer();
    }

    public Beer() {
        //We can assign values to our beer array here.
    }

}
```

- Here, we have our Array structure defined outside of any methods.
- It has a private access level – meaning only the Beer class can access it.

- How can we assign values to our Beer array now?
  With; beer[int] = "String"
  i.e: beer[0] = "Bulmers"
- And how can we now print "Bulmers" from the array?
  System.out.println(beer[0]);

Arrays in Java

- So, now that we have beer[0] = “Bulmers”, how would we retrieve “Bulmers”, and place it into a variable?

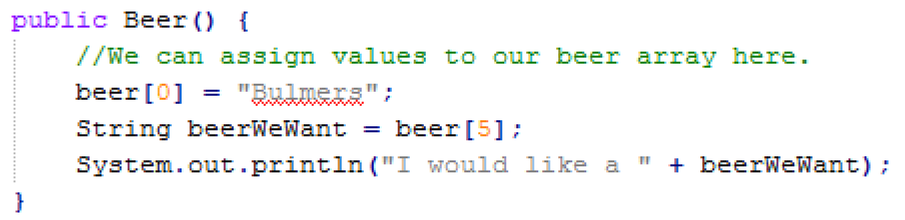


```
public Beer() {
    //We can assign values to our beer array here.
    beer[0] = "Bulmers";
    String beerWeWant = beer[0];
    System.out.println("I would like a " + beerWeWant);
}
```

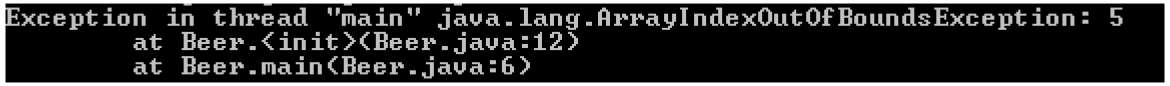
- Simple – we’re working with Strings, so we need to set up a new String Variable.
- We create the new String “beerWeWant”, then assign it beer[0].

Arrays in Java

- So what happens if we do this? (Note: we're trying to access beer[5])



```
public Beer() {
    //We can assign values to our beer array here.
    beer[0] = "Bulmers";
    String beerWeWant = beer[5];
    System.out.println("I would like a " + beerWeWant);
}
```



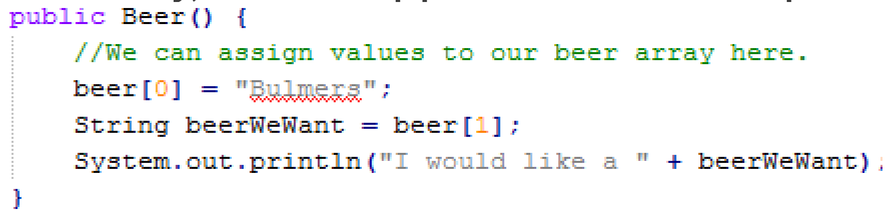
```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at Beer.<init>(Beer.java:12)
        at Beer.main(Beer.java:6)
```

- Java will throw a run-time error.
- We're storing 5 Strings – but we start counting at 0. So for beer[5], the valid positions are;
  beer[0]
  beer[1]
  beer[2]
  beer[3]
  beer[4]

- beer[5], as far as Java, and you are concerned – does not exist.

Arrays in Java

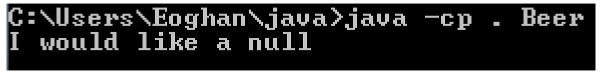
- Similarly, what happens if we attempt to do this?



```java
public Beer() {
    //We can assign values to our beer array here.
    beer[0] = "Bulmers";
    String beerWeWant = beer[1];
    System.out.println("I would like a " + beerWeWant);
}
```

- Here, we're trying to access beer[1] – it's within our array bounds.. But has nothing set to it.

- Java is more forgiving in this circumstance.



```
C:\Users\Eoghan\java>java -cp . Beer
I would like a null
```

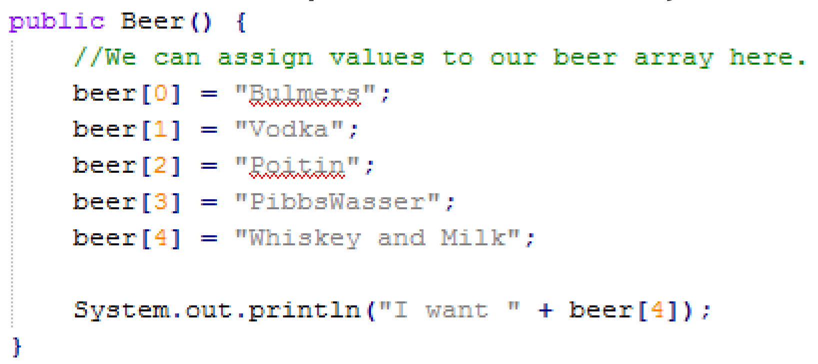
It instead assumes that you want a null String returned.

- When Java initialises this String array, it automatically sets everything to the null String.

- This means that if we try to access an array position we have not yet set, we won't receive a run-time error; instead, Java will return with the null String.

Arrays in Java

- So, let’s fill up our entire array.



```
public Beer() {
    //We can assign values to our beer array here.
    beer[0] = "Bulmers";
    beer[1] = "Vodka";
    beer[2] = "Poitin";
    beer[3] = "PibbsWasser";
    beer[4] = "Whiskey and Milk";

    System.out.println("I want " + beer[4]);
}
```

- Now, we can access any element of the beer array. In this example, we would print out “I want Whiskey and Milk”.

Arrays – Construction recap

- Arrays are set up with Type any-name = new Type[size-of-array];
- We start counting at 0 – so an array set up with size 5, will have positions 0-4.
- Arrays declared as holding Strings, cannot hold other types such as Ints.
- We can access any element of the array with array-name[key]
- The key is ALWAYS an int.
- We can replace the String at any key, at any time; i.e;

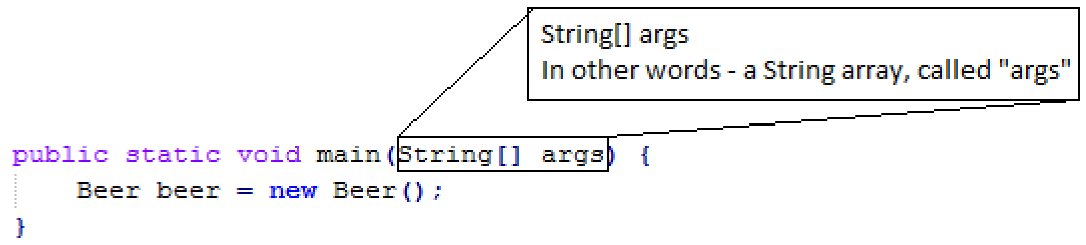
```
beer[0] = “Bulmers”;
System.out.println(beer[0]); // Prints “Bulmers”
beer[0] = “Bloody Mary”;     // beer[0] now holds “Bloody Mary” instead of “Bulmers”
System.out.println(beer[0]); // Prints “Bloody Mary”
```

- Arrays can be declared within methods – but in our example, we declared it outside of a method, so that we could use it throughout our entire program.
- As always, Arrays have access levels; i.e: private, public, etc. Generally, they will be private.

Arrays – Pointers, methods, and better construction

- So far, we've created pretty weak arrays – we're always working with the assumption of knowing EXACTLY how big they need to be beforehand, and knowing EXACTLY where everything is.
- So let's have a look at a new problem – we have no idea how big the array will be.

- All of your Java applications have actually started with an array – it takes in arguments from the command line, when you're launching your application. It's contained within your main method;



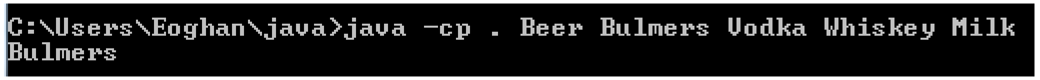
String[] args
In other words - a String array, called "args"

```
public static void main(String[] args) {
    Beer beer = new Beer();
}
```

- So, we have no way to tell how many arguments someone enters at the command line – so say we wanted to take half of these arguments, put them in a new array, and then print them all out. How would we go about doing this?

Arrays – Pointers, methods, and better construction

- So, firstly, here's how command line arguments work; we'll use our Beer class to demonstrate this. Let's say we wanted to pass in the names of every beer we want to sell.
- At the command line, instead of entering "java Beer", we would enter; "java Beer Bulmers Vodka Whiskey Milk"
- This will populate our "args" array, as follows;
  args[0] = "Bulmers"
  args[1] = "Vodka"
  args[2] = "Whiskey"
  args[3] = "Milk"
- So now, if we run the following command;
  System.out.println(args[0]);
  What will be printed out?



```
C:\Users\Eoghan\java>java -cp . Beer Bulmers Vodka Whiskey Milk
Bulmers
```

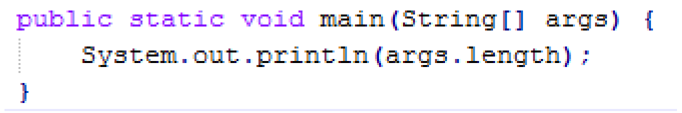
- args[0] contains "Bulmers" – so we'll output "Bulmers".

Arrays – Pointers, methods, and better construction

- Now, we'll need to look at one of the most important methods for Arrays – Length.

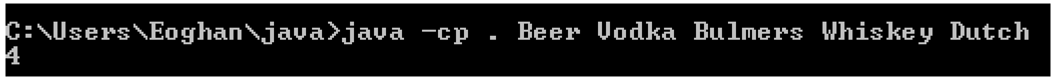
- In the previous example, we've no way to actually tell how big the args[] array is – the user can type in any amount of beers that they want. We can make assumptions that something might be at args[0] – it's likely that there's at least one beer. But how can we tell exactly how much input the user entered?

- All arrays have a public int held within them – called Length. This can be accessed to see exactly how many items are being held inside your array. To access it, simply use;
  array-name.length;
  (or in our case – we're using the args array, so we'll use args.length);



```
public static void main(String[] args) {
    System.out.println(args.length);
}
```

- We'll input the names of 4 drinks – and then our main method will print out the length.



```
C:\Users\Eoghan\java>java -cp . Beer Vodka Bulmers Whiskey Dutch
4
```

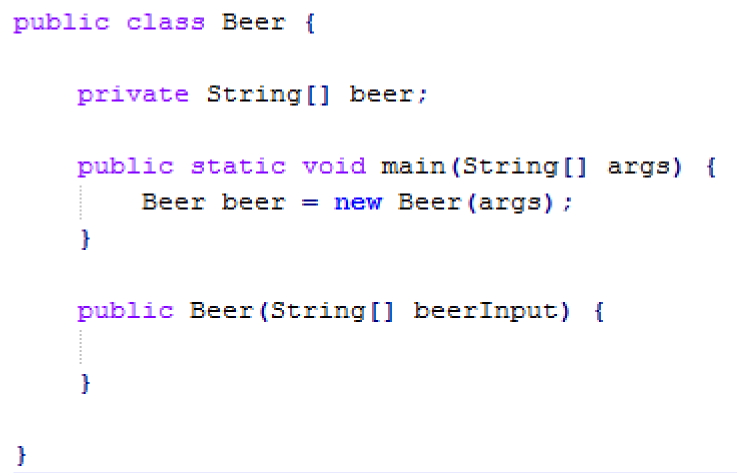
Arrays – Pointers, methods, and better construction

- So, if we want to make a new array, that will take in half the inputs of the args array – how will we go about doing that?

- Firstly, remember our problem with static variables and objects – we want to do this outside of the main method.

- So, we need to do two things, which we saw in our Objects revision;
  1) Set up the array (without initialising it), outside of any methods.
  2) Send our 'args' array into the Beer constructor

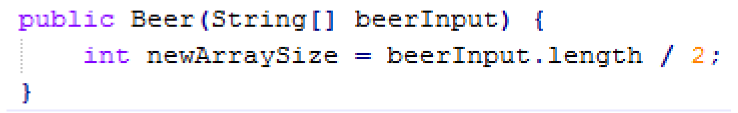
  And from here, we can then start working on the logic of our code.



```java
public class Beer {

    private String[] beer;

    public static void main(String[] args) {
        Beer beer = new Beer(args);
    }

    public Beer(String[] beerInput) {

    }

}
```

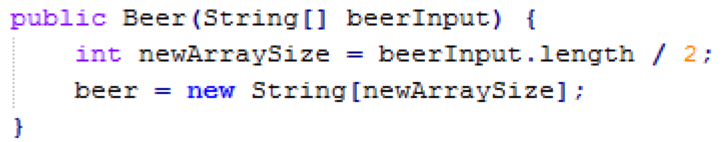
Arrays – Pointers, methods, and better construction

- So, we can use array.length to find out how large the array is – we want half of that size.



```
public Beer(String[] beerInput) {
    int newArraySize = beerInput.length / 2;
}
```

- Now, we have half the array size – so let's make a new array of that size.
  Remember that we've already declared the array to be "beer" – but we have not initialised it.



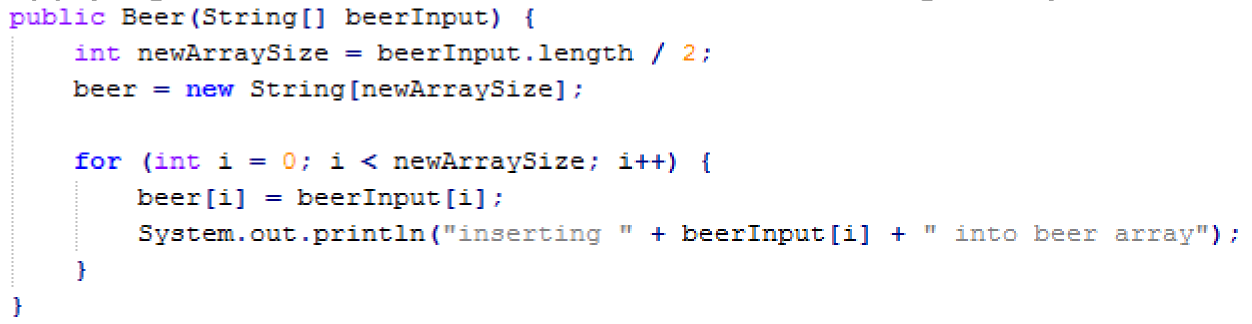
```
public Beer(String[] beerInput) {
    int newArraySize = beerInput.length / 2;
    beer = new String[newArraySize];
}
```

- So, how can we go about transferring the first 2 elements of the args array, into our beer array?
- We can create a pointer to do this for us. In Java, we can consider a pointer to simply be an integer, whose sole purpose is to point at the array element that we want to access.

  For example;
  int arrayPointer = 0;
  System.out.println(beer[arrayPointer]); // Prints the beer at position 0
  arrayPointer++;
  System.out.println(beer[arrayPointer]); // Prints the beer at position 1
- Warning: This is not the standard definition of a pointer – don't confuse it with pointers as you've seen in NASM!

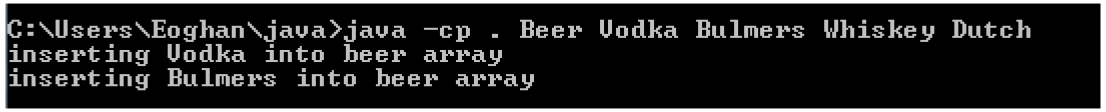
Arrays – Pointers, methods, and better construction

- So, we can set up a pointer – and in a for loop, we can iterate through half of the args array, and applying the values from the first half of the args array into our new beer array.



```
public Beer(String[] beerInput) {
    int newArraySize = beerInput.length / 2;
    beer = new String[newArraySize];

    for (int i = 0; i < newArraySize; i++) {
        beer[i] = beerInput[i];
        System.out.println("inserting " + beerInput[i] + " into beer array");
    }
}
```

- Giving us this output;



```
C:\Users\Eoghan\java>java -cp . Beer Vodka Bulmers Whiskey Dutch
inserting Vodka into beer array
inserting Bulmers into beer array
```

- Here, we used the variable ‘i’ as a pointer – in short, the code executed above was;
  beer[0] = beerInput[0];
  beer[1] = beerInput[1];

  Our newArraySize was set to 2, and this for loop will only run while i is less than 2;

  So i will be equal to 0, then carry out the code in the loop – then i will be incremented to 1, and run through the loop again. Once i is equal to 2, the for loop will end.

Arrays – Pointers, methods, and better construction

- So with this logic, we can actually run through an array of any size, using the following;

```
for (int pointer = 0; pointer < array.length; pointer++) {
        //Print out every String in this array;
        System.out.println(array[pointer]);
}
```

- Similarily, we can use the following logic to run through an array, from back to front;

```
for (int pointer = (array.length – 1); pointer <= 0; pointer--) {
         //Print out every String in this array;
        System.out.println(array[pointer]);
}
```

- This is just simple pointer manipulation – practice it, as you'll be using it forever.

Arrays – Pointers, methods, and better construction

- Finally, let's take a look at one other way to iterate through an array; an enhanced for loop.
- This time, we'll go from start to finish, without having to initialise a pointer.

- Enhanced for loop structure;

```
for (Type any-name-you-want : array-name) {
        System.out.println(any-name-you-want);
}
```

- So, for our beer array;

```
for (String beerName : beer) {
        System.out.println(beerName);
}
```

- This will iterate through EVERYTHING in your array. This can be used for any array;
    - Example: for (String argument : args), will allow you to iterate through the args array, in main().

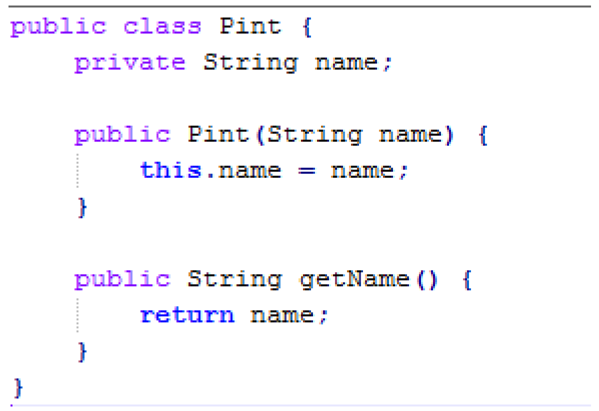
Arrays – Pointers, methods, and better construction

- If you've been paying much attention in your labs and lectures, you'll (hopefully) know that a String is not a primitive type, like an int – it is infact an Object (as referenced by it's capital letter).

  So, what's to stop you from making arrays of Objects?

- Nothing. It's one of the best ways to store Objects of the same type.

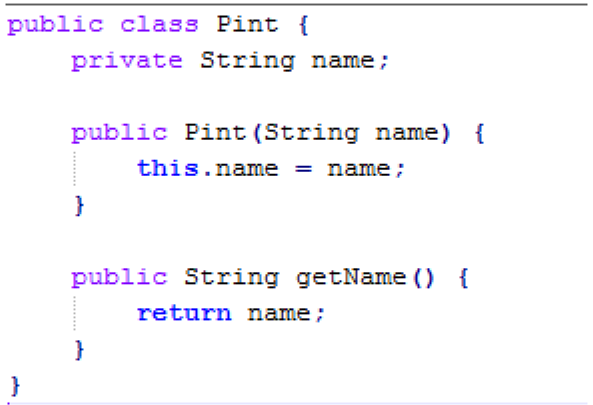
- Let's say we have a pint object – and it's constructor takes in, and stores the name of a drink. It contains a method to return the name of this drink, when requested.



```
public class Pint {
    private String name;

    public Pint(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

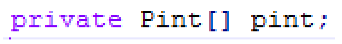
- How would we go about taking input from the command line, and creating a new object for every drink name given?

Arrays – Pointers, methods, and better construction



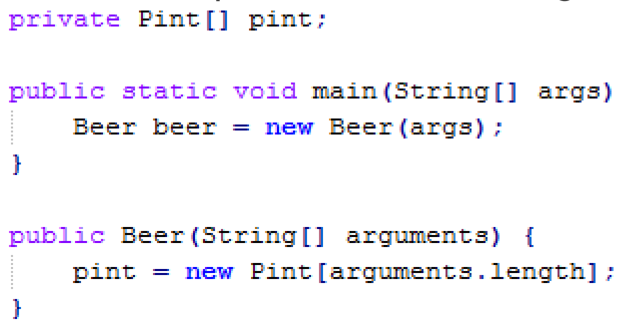
```
public class Pint {
    private String name;

    public Pint(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

- How would we go about taking input from the command line, and creating a new object for every drink name given?
- We create a new array, to handle the Type 'Pint' – we do not yet know it's length.



```
private Pint[] pint;
```

- The user's input is taken in through the String[] args array in main – so we can find out length from this.

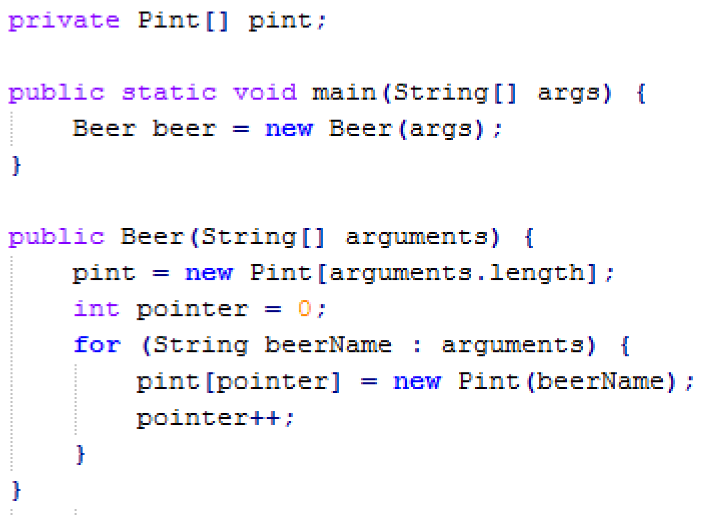


```
private Pint[] pint;

public static void main(String[] args) {
    Beer beer = new Beer(args);
}

public Beer(String[] arguments) {
    pint = new Pint[arguments.length];
}
```

- Now, we can use our enhanced for loop, and for loop iterated by a pointer, to create a new pint object, based on every argument given – and store it in our pint array.

Arrays – Pointers, methods, and better construction

- So, again:
  - Empty Pint[] array, called pint
  - All user input being sent into the args array, in our main method
  - args array is being carried into our constructor – and then referred to as “arguments”.
  - Pint array being created with the same size as the arguments array
  - An enhanced for loop will run through EVERY String in the arguments array
  - We’ll then use a pointer to run through, and fill everything in out pint array, with Pint objects.

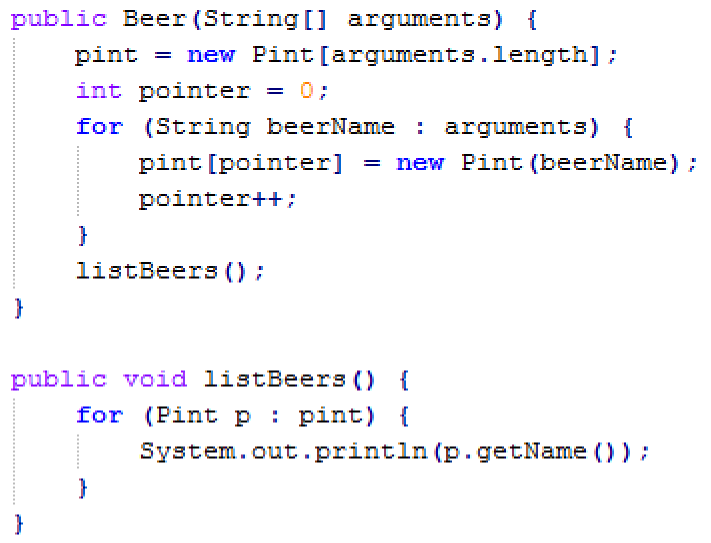


```
private Pint[] pint;

public static void main(String[] args) {
    Beer beer = new Beer(args);
}

public Beer(String[] arguments) {
    pint = new Pint[arguments.length];
    int pointer = 0;
    for (String beerName : arguments) {
        pint[pointer] = new Pint(beerName);
        pointer++;
    }
}
```

- And finally, we can create a quick method that will print out all of our beers, with another enhanced for loop.

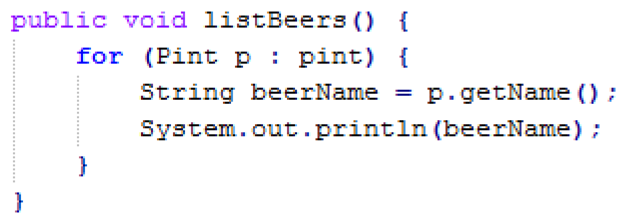
Arrays – Pointers, methods, and better construction

- So we'll make a method called listBeers – which takes in no arguments, and calls the getName(); method on every Pint object in our array.



```
public Beer(String[] arguments) {
    pint = new Pint[arguments.length];
    int pointer = 0;
    for (String beerName : arguments) {
        pint[pointer] = new Pint(beerName);
        pointer++;
    }
    listBeers();
}

public void listBeers() {
    for (Pint p : pint) {
        System.out.println(p.getName());
    }
}
```

- Similarly, we could use;



```
public void listBeers() {
    for (Pint p : pint) {
        String beerName = p.getName();
        System.out.println(beerName);
    }
}
```

- Now, we have a program that interacts with command line input, sets up an array of custom objects, and uses both pointers, and an enhanced for loop to run through them.

Arrays – Pointers, methods, and better construction

- Is it a headache to remember all this?
  To start with, yes, it can be. Practice it until it's second nature – Arrays are very simple once you get used to them.

- Can we go back to using associative arrays as we did with PHP?
  You'll see similar again in Dr. Herley's course on Linear Data Structures.
  For those of you inclined, look up ArrayLists – there are tonnes of tutorials available online.

- Is there a way to force Java to let me use more than one type in an array?
  Yes, by using generics – you'll see these later in the year. But you should not do this without fantastic reason.

- I'm still confused about Arrays
  This is a large topic, and a huge fundamental for all modern programming languages.

  Read through the slides, study the course material provided by Dr. Marc Van Dongen.

  Do not stop studying arrays until you are 100% confident with them;
  knowing your data types can, and will save your exam results.

- Recommended practice:
  Start with non-object based arrays; use static everywhere if you need to, for practice sake.
  Start with basic, known types, such as Strings, or integers.
  Use command line arguments (or the Scanner object) to fill these arrays.