Parameters of Different Types

Using Primitive, Object, and Array Parameters

Alwin Tareen

Parameters of Primitive Types

- When a method has a primitive type as a parameter, Java passes that parameter using a technique called **pass-by-value**.
- A method that is passed a parameter of a primitive type is **passed a copy** of that parameter's value.
- This means that the calling method has no access to this parameter itself.

Pass-by-Value

```
public class PassingPrimitives
{
    public static void displayScore()
    {
        int score = 10;
        displayTotal(score);
        System.out.println(score);
    }
    public static void displayTotal(int total)
    {
        total = 75;
        System.out.println(total);
    }
    public static void main(String[] args)
    {
        displayScore();
    }
}
```

Pass-by-Value

- `displayScore()` begins by defining a local variable of type `int` named `score`, and initializing it with a value of `10`.

Java Memory

`score`	10

- A call is made to the `displayTotal()` method, and `score` is passed as a parameter.
- The method `displayTotal()` executes, and a copy of the value in `score` is assigned to `total`.

Java Memory

`score`	10
`total`	10

Pass-by-Value

- `total` is then assigned a value of `75`. Notice that this has no effect on the variable `score`.

Java Memory

`score`	10
`total`	75

- `total` is then displayed.
- Execution then returns to `displayScore()`, where `score` is displayed. Since `total` received a copy of `score`'s value, `displayTotal()` had no access to `score`.

Java Memory

`score`	10

Parameters of Object Types

- When a method has an object type as a parameter, Java passes that parameter using a technique called **pass-by-reference**.
- A method that is passed a parameter of an object type is passed the reference(memory location) to the object assigned to that parameter.

Parameters of Object Types

```
public class Digit
{
    private int num;

    public Digit()
    {
        num = 0;
    }

    public int getNum()
    {
        return num;
    }

    public void setNum(int n)
    {
        num = n;
    }
}
```

Parameters of Object Types

```
public class PassingObjects
{
    public static void outputResult()
    {
        Digit item = new Digit();
        outputSolution(item);
        System.out.println(item.getNum());
    }
    public static void outputSolution(Digit element)
    {
        element.setNum(75);
        System.out.println(element.getNum());
    }
    public static void main(String[] args)
    {
        outputResult();
    }
}
```

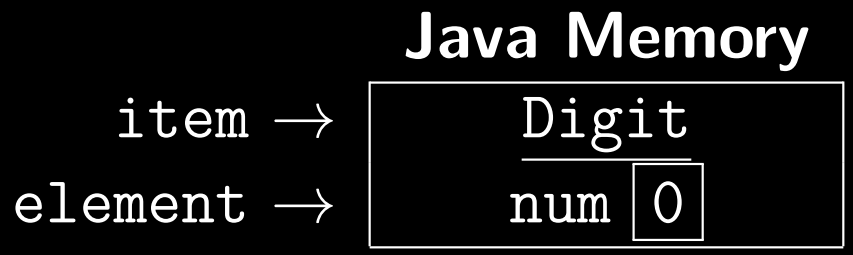
Parameters of Object Types

- `outputResult()` begins by instantiating(creating) a `Digit` object and storing its reference in a variable named `item`.

- A call is made to the `outputSolution()` method, and `item` is passed as a parameter.

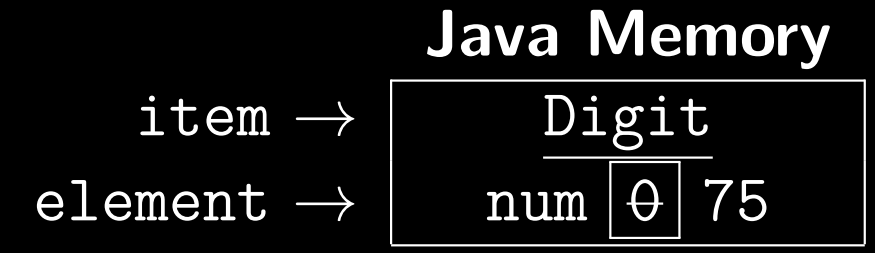
Parameters of Object Types

- Inside `outputSolution()`, `element` is passed a reference to the object assigned to `item`. This means that `item` and `element` are now referencing the same object. A copy of the object is not made(as was done when using pass-by-value).



Parameters of Object Types

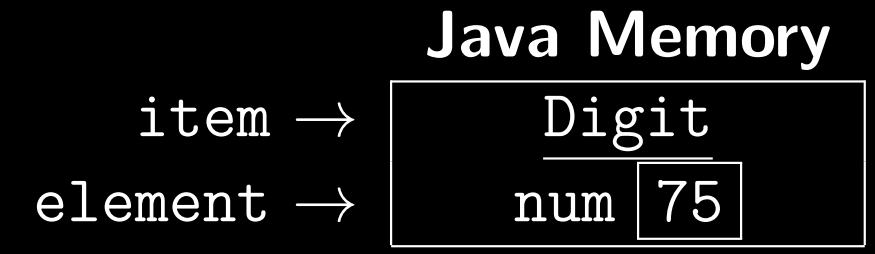
- Then, the `Digit` class' `setNum()` method is executed on the `element` object. This updates the instance variable `num`, giving it a value of `75`.



- Then, the `Digit` class' `getNum()` method is run on the `element` object, which displays the value of `num`.

Parameters of Object Types

- Execution returns to `outputResult()` where a call is made to the `Digit` class' `getNum()` method on the `item` object. The value of `num` is displayed: `75`.



Parameters of an Array Type

- Arrays are objects, and they can be passed as parameters to other methods.
- This means that the arrays are passed using **pass-by-reference**.
- In other words, the contents of the array can be altered by the method to which it was passed.

The PassingArrays Class

```
public class PassingArrays
{
    public static void initializeArray()
    {
        int[] scores = {10, 20, 30};
        alterArray(scores);
        displayArray(scores);
    }

    public static void alterArray(int[] points)
    {
        displayArray(points);
        points[0] = 500;
    }
```

The PassingArrays Class, Continued

```
    public static void displayArray(int[] arr)
    {
        for (int i = 0; i < arr.length; i++)
        {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args)
    {
        initializeArray();
    }
}
```

Parameters: End of Notes