CSIS-1400

Final Prep

Scope

- Sope of a local variable

Local Variables

- Where can they be declared?
- Are they default initialized?
- When do they need to be initialized?

Primitive Types

- How many primitive types are there?
- List primitive types

```
double x = 7;
```

Check all statements that divide x in half:

a) `x /= 2;`

b) `x = x / x;`

c) `x -= x / 2;`

d) `x *= 1 / 2;`

e) `x = (x > x / 2) ? x / 2 : 1 / 2 * x;`

```
double x = 7;
```

Check all statements that divide x in half:

[✓ a) `x /= 2;`]

b) `x = x / x;`

[✓ c) `x -= x / 2;`]

d) `x *= 1 / 2;`

[✓ e) `x = (x > x / 2) ? x / 2 : 1 / 2 * x;`]

Array of type double:

```
double[] quizResults = new double[7];
```

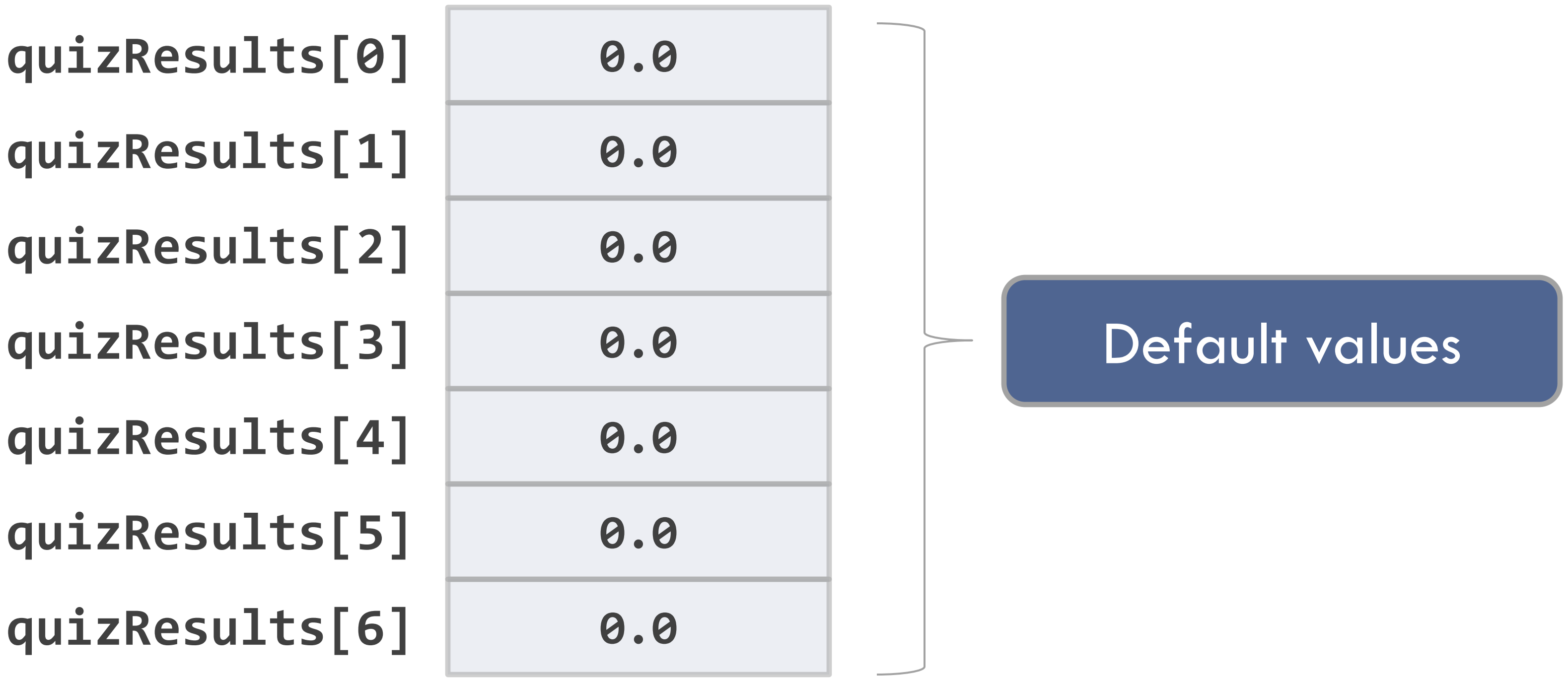
Example:

```
double[] quizResults = new double[7];
```

`quizResults[0]`	0.0
`quizResults[1]`	0.0
`quizResults[2]`	0.0
`quizResults[3]`	0.0
`quizResults[4]`	0.0
`quizResults[5]`	0.0
`quizResults[6]`	0.0

Example:

```
double[] quizResults = new double[7];
```



Example:

```
double[] quizResults = new double[7];
```

`quizResults[0]`	`0.0`
`quizResults[1]`	`0.0`
`quizResults[2]`	`0.0`
`quizResults[3]`	`0.0`
`quizResults[4]`	`0.0`
`quizResults[5]`	`0.0`
`quizResults[6]`	`0.0`

Indices: from 0 to quizResults.length - 1

Example:

```
double[] quizResults = new double[7];

quizResults[0] = 89.5;

quizResults[2] = 94;

quizResults[4] = 92.5;
```

Example:

```
double[] quizResults = new double[7];
```

`quizResults[0]`	89.5
`quizResults[1]`	0.0
`quizResults[2]`	94
`quizResults[3]`	0.0
`quizResults[4]`	92.5
`quizResults[5]`	0.0
`quizResults[6]`	0.0

Example:

```
double[] quizResults = new double[7];
```

`quizResults[0]`	89.5
`quizResults[1]`	0.0
`quizResults[2]`	94
`quizResults[3]`	0.0
`quizResults[4]`	92.5
`quizResults[5]`	0.0
`quizResults[6]`	0.0

Elements of a value-type array include the actual value (e.g. 89.5)

Array of type string:

```
String[] names = new String[4];
```

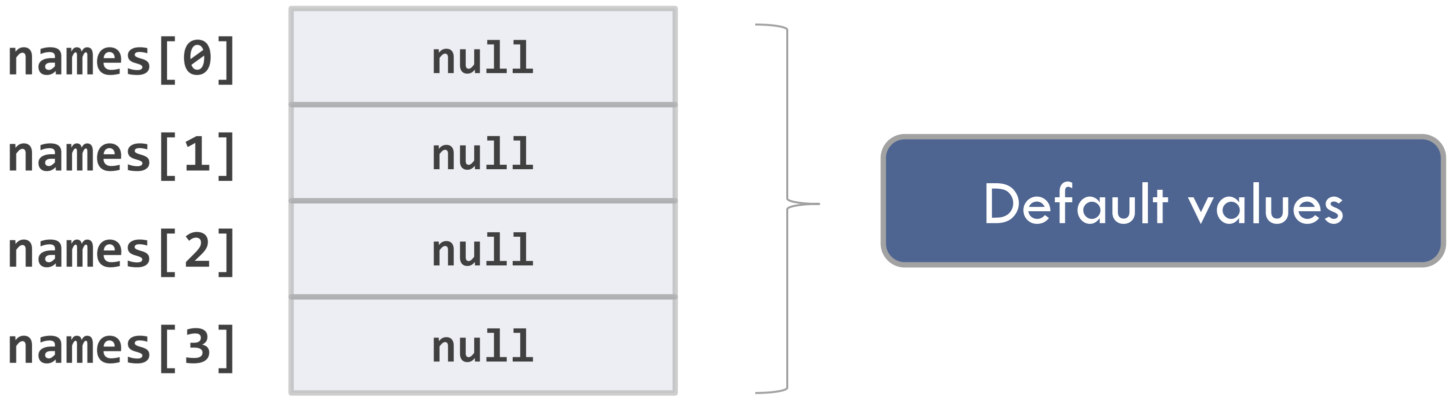
Example:

```
String[] names = new String[4];
```

`names[0]`	`null`
`names[1]`	`null`
`names[2]`	`null`
`names[3]`	`null`

Example:

```
String[] names = new String[4];
```



Example:

```
String[] names = new String[4];
```

`names[0]`	`null`
`names[1]`	`null`
`names[2]`	`null`
`names[3]`	`null`

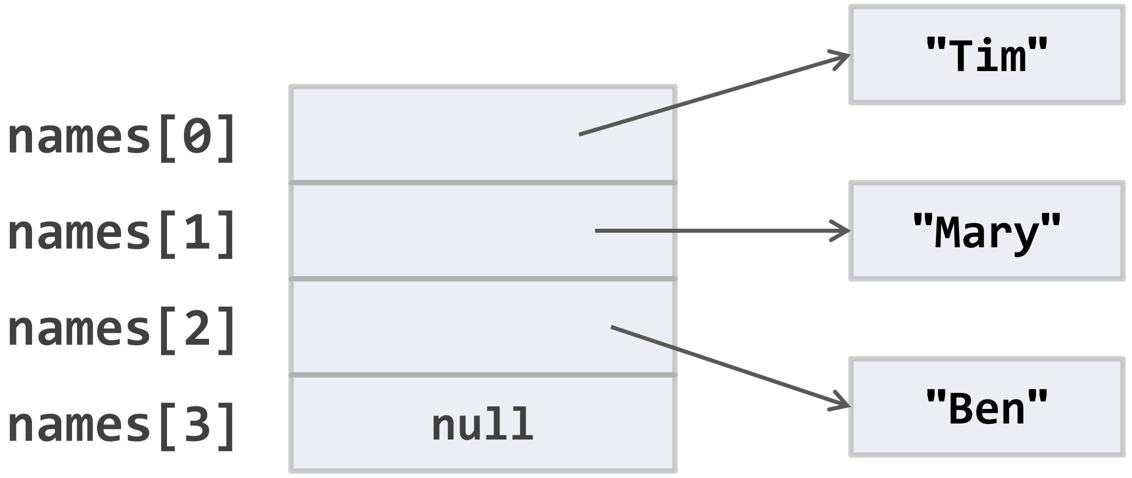
Indices: from 0 to names.length - 1

Example:

```
String[] names = new String[4];
names[0] = "Tim";
names[1] = "Mary";
names[2] = "Ben";
```

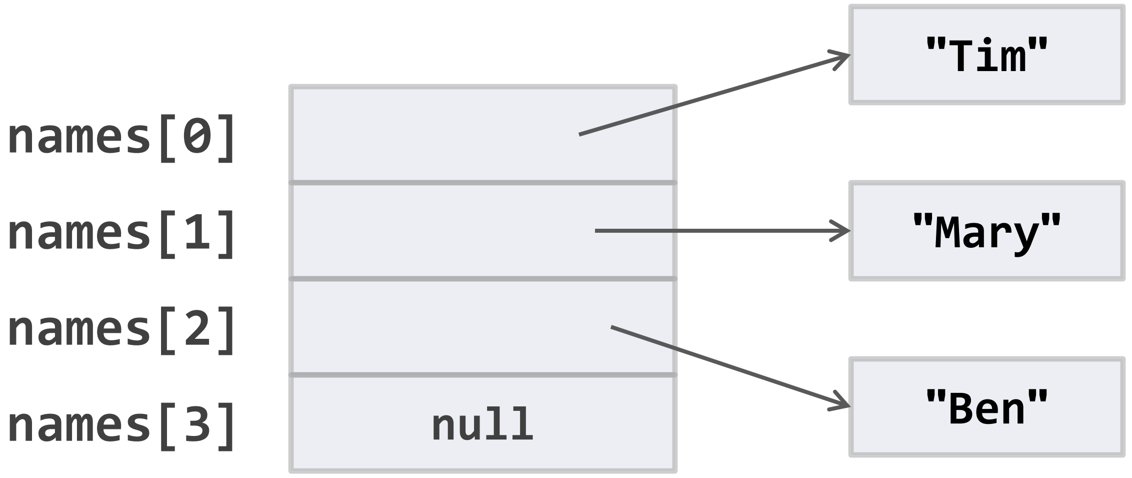
Example:

```
String[] names = new String[4];
```



Example:

```
String[] names = new String[4];
```



Elements of a reference-type array include a reference to the actual object

TODO: Arrays

How do you declare an array variable?

How do you access an array element?

What is the first / last index of array myArray?

How do you pass an array argument to a method?

Can an array be resized (yes / no) ?

TODO: Arrays

How do you declare an array variable?

```
type[] myArray;        e.g. int[] iArray;
```

How do you access an array element?

What is the first / last index of array myArray?

How do you pass an array argument to a method?

Can an array be resized (yes / no) ?

TODO: Arrays

How do you declare an array variable?

```
type[] myArray;         e.g. int[] iArray;
```

How do you access an array element?

```
myArray[index]          e.g. iArray[3]
```

What is the first / last index of array myArray?

How do you pass an array argument to a method?

Can an array be resized (yes / no) ?

TODO: Arrays

How do you declare an array variable?

```
type[] myArray;          e.g. int[] iArray;
```

How do you access an array element?

```
myArray[index]          e.g. iArray[3]
```

What is the first / last index of array myArray?

```
first index: 0      last index: myArray.length – 1
```

How do you pass an array argument to a method?

Can an array be resized (yes / no) ?

TODO: Arrays

How do you declare an array variable?

```
type[] myArray;        e.g. int[] iArray;
```

How do you access an array element?

```
myArray[index]         e.g. iArray[3]
```

What is the first / last index of array myArray?

```
first index: 0      last index: myArray.length – 1
```

How do you pass an array argument to a method?

```
pass the name          e.g. myMethod(myArray)
```

Can an array be resized (yes / no) ?

TODO: Arrays

How do you declare an array variable?

`type[] myArray;`   e.g. `int[] iArray;`

How do you access an array element?

`myArray[index]`   e.g. `iArray[3]`

What is the first / last index of array myArray?

`first index: 0`   `last index: myArray.length – 1`

How do you pass an array argument to a method?

`pass the name`   e.g. `myMethod(myArray)`

Can an array be resized (yes / no) ?

`NO, it can not be resized once it has been created`

Arrays:

How do you use an array initializer?

Arrays:

How do you use an array initializer?

List comma separated array elements in curly braces

e.g.:

```
int[] iArray = {10, 20, 30, 40};

String[] strArray = {“Tim”, “Don”, “Pat”};

Point[] ptArray = {new Poit(2,3), new Point(4, 1)};
```

EXERCISE 1

TODO:

Circle all expressions that subtract 1 of the array element on index i?

- `--arrayName[i]`
- `arrayName--[i]`
- `arrayName[i]--`
- `arrayName[i--]`
- `arrayName[--i]`

Circle all expressions that subtract 1 of the array element on index i?

- [✓ `--arrayName[i]`]
- `arrayName--[i]`
- [✓ `arrayName[i]--`]
- `arrayName[i--]`
- `arrayName[--i]`

Circle all expressions that subtract 1 of the array element on index i?

- `--arrayName[i]`
- `arrayName--[i]`
- `arrayName[i]--`
- `arrayName[i--]`
- `arrayName[--i]`

The array name is immediately followed by the index

Circle all expressions that subtract 1 of the array element on index i?

- `--arrayName[i]`
- `arrayName--[i]`
- `arrayName[i]--`
- `arrayName[i--]`
- `arrayName[--i]`

The array name is immediately followed by the index

Here the index itself is decremented, not the element on that index

Enhanced `for` Statement **foreach loop**

- Syntax:

```
for ( type name : arrayNames )
{
    statement(name)
}
```

arrayName … name of array through which we iterate.

type … must match array’s element type.

Enhanced `for` Statement (for-each loop)

Pro:

- Clear and concise syntax
  - makes code easier to read
- Robust
  - Avoids off-by-1 errors

Enhanced `for` Statement (for-each loop)

Con:

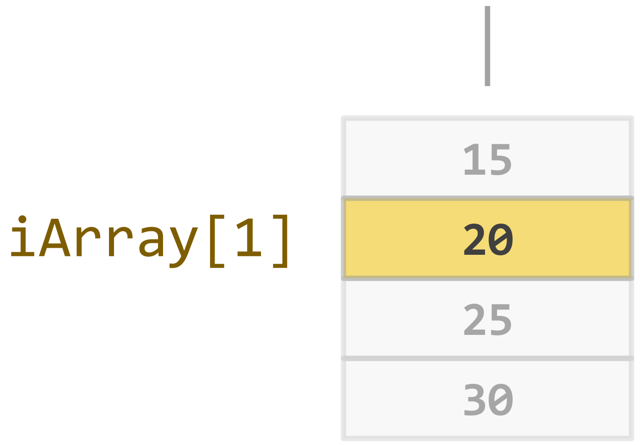
- Can't re-assign elements
- Provides no index

If you need to assign new elements or access the index use the counter controlled for loop

EXERCISE 2

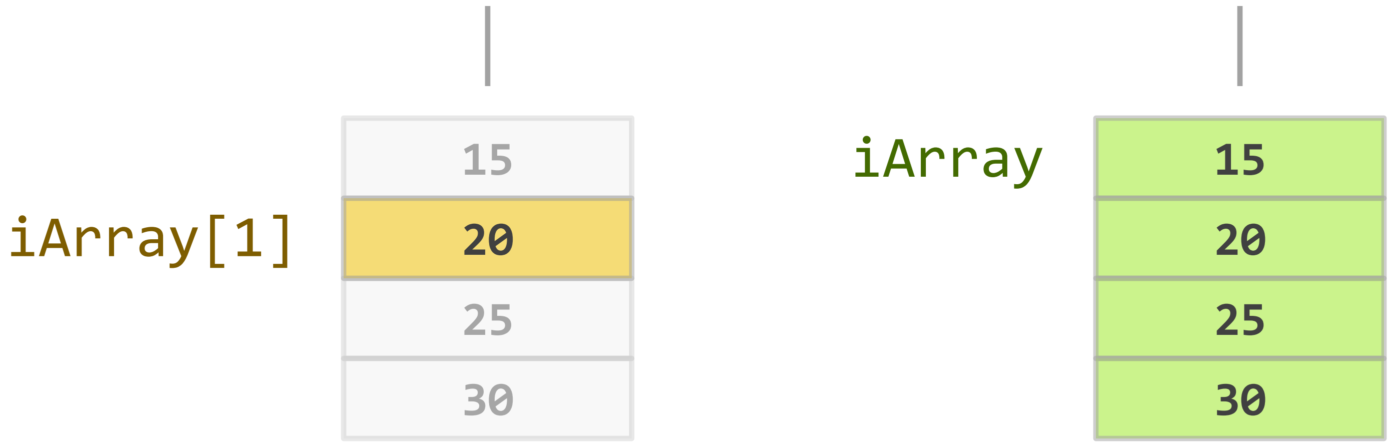
Passing Arrays to Methods

- You can pass single array elements or the whole array



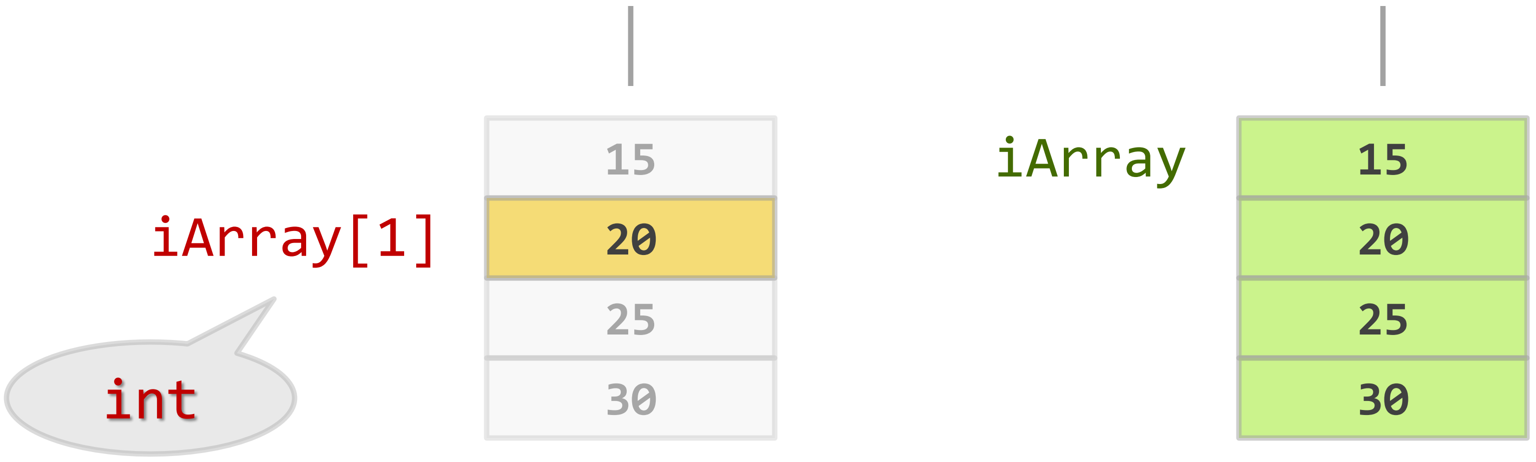
7.7 Passing Arrays to Methods

- You can pass single array elements or the whole array



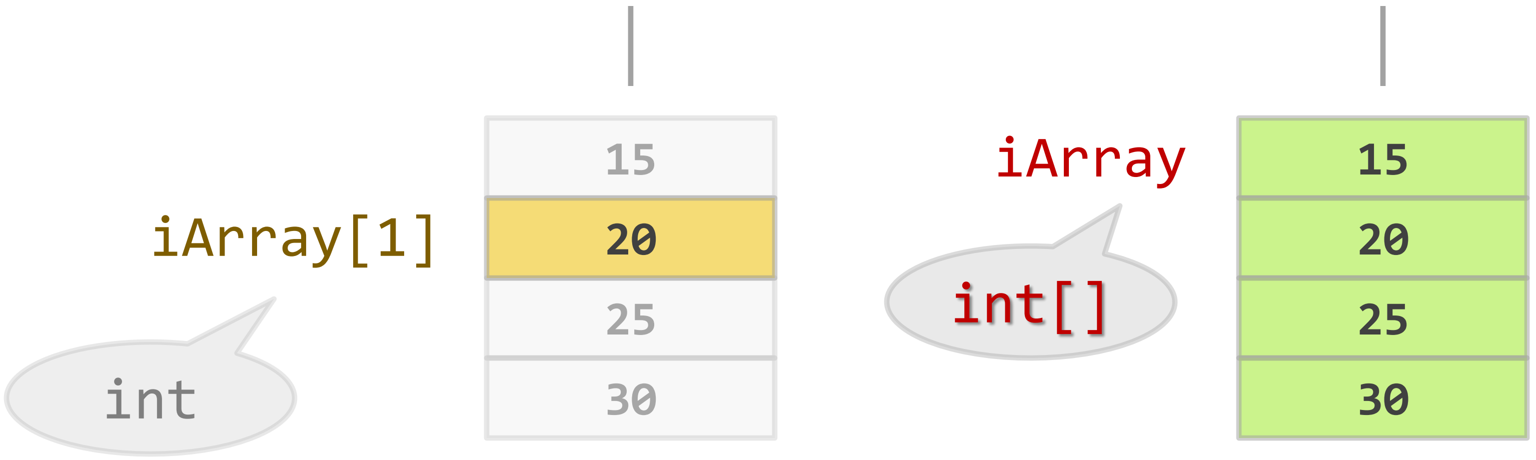
7.7 Passing Arrays to Methods

- You can pass single array elements or the whole array



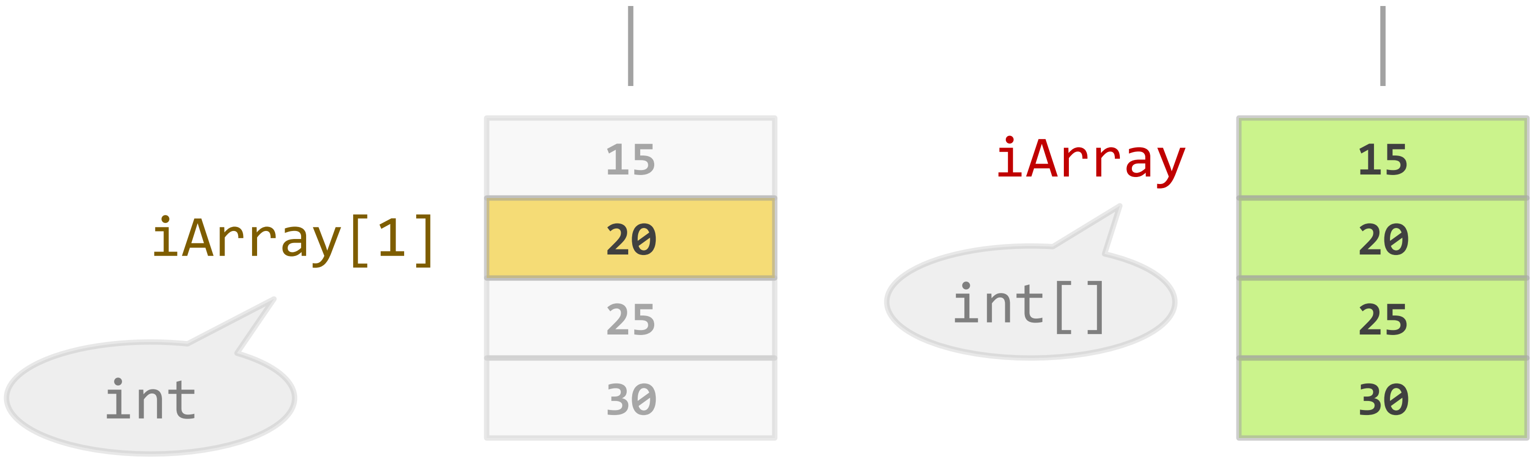
7.7 Passing Arrays to Methods

- You can pass single array elements or the whole array



7.7 Passing Arrays to Methods

- You can pass single array elements or the whole array



Whatever you pass as an argument needs to match the parameter type in the method declaration

Passing array elements:

Variable declaration:

```
int[] iArray = { 2, 4, 6, 8 };
```

Method declaration:

```
void printTriangle (int size)
{
   // print triangle
}
```

Passing array elements:

Variable declaration:

```
int[] iArray = { 2, 4, 6, 8 };
```

Method declaration:

```
void printTriangle (int size)
{
   // print triangle
}
```

Method call:

```
printTriangle(iArray[1]);
```

Passing array elements:

Variable declaration:

```
int[] iArray = { 2, 4, 6, 8 };
```

Method declaration:

```
void printTriangle (int size)
{
   // print triangle
}
```

`int` is no longer mentioned, but whatever is passed has to be of type int

Method call:

```
printTriangle(iArray[1]);
```

Passing arrays:

Variable declaration:

```
int[] iArray = { 2, 4, 6, 8 };
```

Method declaration:

```
void printArray (int[] iArray)
{
    for(int el : iArray)
   {
        System.out.printf(“%d “, el);
   }
}
```

Passing arrays:

Variable declaration:

```
int[] iArray = { 2, 4, 6, 8 };
```

Method declaration:

```
void printArray (int[] iArray)
{
    for(int el : iArray)
   {
        System.out.printf("%d ", el);
   }
}
```

Method call:

```
printArray(iArray);
```

Passing arrays:

Variable declaration:

```
int[] iArray = { 2, 4, 6, 8 };
```

Method declaration:

```
void printArray (int[] iArray)
{
    for(int el : iArray)
   {
        System.out.printf("%d ", el);
   }
}
```

int[] is no longer mentioned, but whatever is passed has to be of type int[]

Method call:

```
printArray(iArray);
```

Passing arrays:

Variable declaration:

```
int[] iArray = { 2, 4, 6, 8 };
```

Method declaration:

```
void printArray (int[] iArray)
{
    for(int el : iArray)
   {
        System.out.printf("%d ", el);
   }
}
```

`int[]` is no longer mentioned, but whatever is passed has to be of type int[]

Method call:

```
printArray(iArray);
```

You pass the name or the array

T / T[ ] / ArrayList<T>

e.g.

String

String[]

ArrayList<String>

EXERCISE 3a

ArrayList vs Array

ArrayList	Array
can grow and shrink	fixed size
reference types only	primitive and referenct types
	very efficient

Be prepared for an exercise where you have to write some code using methods from class Arrays and/or ArrayList<E>

Class Arrays

Class ArrayList<E>

Example 4:

Code reading: ArrayList or array:

Print the output produced by the method below. Pay attention to details like new lines etc.

```
ArrayList<String> treeList = new ArrayList<String>();
 treeList.add("oak");
 treeList.add("ash");
 treeList.add("fir");

 System.out.println(treeList);

 treeList.add(2, "banyan");
 treeList.remove("oak");

 for (String tree: treeList)
 {
     System.out.printf("%s ", tree);
 }
```

Example 4:

Code reading: ArrayList or array:

Print the output produced by the method below. Pay attention to details like new lines etc.

```
ArrayList<String> treeList = new ArrayList<String>();
treeList.add("oak");
treeList.add("ash");
treeList.add("fir");

System.out.println(treeList);


treeList.add(2, "banyan");
treeList.remove("oak");

for (String tree: treeList)
{
    System.out.printf("%s ", tree);
}
```

[oak, ash, fir]
ash banyan fir

Method / Ctor Overloading

- What is method / constructor overloading?
- Why do people overload methods / constructors?

Enum rules and restrictions:

- All enum types are reference types.
- enum declaration: comma-separated list of enum constants
- `enum` constants are implicitly final
- `enum` constants are implicitly static.
- static method values returns array of the enum’s constants.

```
enum Season { WINTER, SPRING, SUMMER, FALL }
```

implicitly static and final

All Uppercase

```
enum Season { WINTER, SPRING, SUMMER, FALL }
```

implicitly static and final

All Uppercase

NO double quotes

```
enum Season { WINTER, SPRING, SUMMER, FALL }
```

implicitly static and final

Java API

Match the functionality provided to one of those 4 types

- Scanner
- Math
- Arrays
- ArrayList

Example 5

Random **A**	I need to simulate tossing a coin 2000 times
	I need to calculate the square root of 15
Math **B**	I need to print the first 10 digits of pi
	I need to read in the title of the missing book
Scanner **C**	I'd like to set the value of all elements in my byte array to 6
	It returns a value between 25 and 100
Arrays **D**	I need to sort an all the points in a given array
	I need the user to enter his gpa

Exercise 5

Random **A**	I need to simulate tossing a coin 2000 times **A**
	I need to calculate the square root of 15 **B**
Math **B**	I need to print the first 10 digits of pi **B**
	I need to read in the title of the missing book **C**
Scanner **C**	I'd like to set the value of all elements in my byte array to 6 **D**
	It returns a value between 25 and 100 **A**
Arrays **D**	I need to sort an all the points in a given array **D**
	I need the user to enter his gpa **C**

EXERCISE 6

STATIC

How is a static field different from an instance field?

Give an example of a static field / instance field

STATIC

How is a static field different from an instance field?

Give an example of a static field / instance field

How is a static method different from an instance method

Give an example when to use a static method / instance method

There will be examples similar to those on the array quiz, the loop quiz and/or the accessing instance methods quiz

Please review the quiz prep materials.

EXERCISE 7