Grouping Objects

Primitive Arrays and Iteration

Produced by: Dr. Siobhán Drohan

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

Topic List

- Primitive arrays
  - Why do we need them?
  - What are they?
  - Using a primitive array.
  - Recap: for and while loops.
  - Arrays and counter-controlled loops.
  - Arrays and sentinel-based loop.
  - Arrays and flag-based loops.
  - Do you have to use all elements in the array?

Why arrays?

- We look at different pieces of code to explain the concept.

- In each case:
  - we read in 5 numbers from the keyboard
  - add them
  - print the result.

Adding 5 numbers

- Reads in 5 numbers from the keyboard
- Adds the numbers
- Prints out the result
- Does not remember the numbers

```
public void askForFiveNumbers()
{
    int n;
    int sum = 0;
    for (int i = 0; i<5; i++)
    {
        System.out.print("Please enter a number: ");
        n = scanner.nextInt();
        sum += n;
    }
    System.out.println("The sum of the values you typed in is: " + sum);
}
```

Rule – Never lose data

- Should always try to store that data for later use (in a more real-life system you would almost always need to use the input data again).
- The previous code has not done this.
- We could try another way ...

Remembering the 5 numbers

```
public void askForFiveNumbersAndRemember()
{
    int n1, n2, n3, n4, n5;
    int sum = 0;

    System.out.print("Please enter a number: ");
    n1 = scanner.nextInt();
    System.out.print("Please enter a number: ");
    n2 = scanner.nextInt();
    System.out.print("Please enter a number: ");
    n3 = scanner.nextInt();
    System.out.print("Please enter a number: ");
    n4 = scanner.nextInt();
    System.out.print("Please enter a number: ");
    n5 = scanner.nextInt();
    sum = n1 + n2 + n3 + n4 + n5;

    System.out.println("The sum of the values you typed in is: " + sum);
}
```

Remembering the 5 numbers

```
public void askForFiveNumbersAndRemember()
{
    int n1, n2, n3, n4, n5;
    int sum = 0;

    System.out.print("Please enter a number: ");
    n1 = scanner.nextInt();
    System.out.print("Please enter a number: ");
    n2 = scanner.nextInt();
    System.out.print("Please enter a number: ");
    n3 = scanner.nextInt();
    System.out.print("Please enter a number: ");
    n4 = scanner.nextInt();
    System.out.print("Please enter a number: ");
    n5 = scanner.nextInt();
    sum = n1 + n2 + n3 + n4 + n5;

    System.out.println("The sum of the values you typed in is: " + sum);
}
```

- This works in the sense that we have retained the input data.
- But:
  - We cannot use loops.
  - If we had to read in 1,000 numbers, this would require extensive code.
- We need another, new data structure...
  - enter arrays...

Topic List

- Primitive arrays
  - Why do we need them?
  - What are they?
  - Using a primitive array.
  - Recap: for and while loops.
  - Arrays and counter-controlled loops.
  - Arrays and sentinel-based loop.
  - Arrays and flag-based loops.
  - Do you have to use all elements in the array?

Arrays (fixed-size collections)

- The collections framework (e.g. ArrayList) are flexible-sized collections.
- Sometimes the maximum collection size can be pre-determined.
- Programming languages usually offer a special fixed-size collection type: an *array*.
- Java arrays can store objects or primitive-type values.
- Arrays use a special syntax.

Single box

If you think of a variable (field, local variable) as a box in memory:

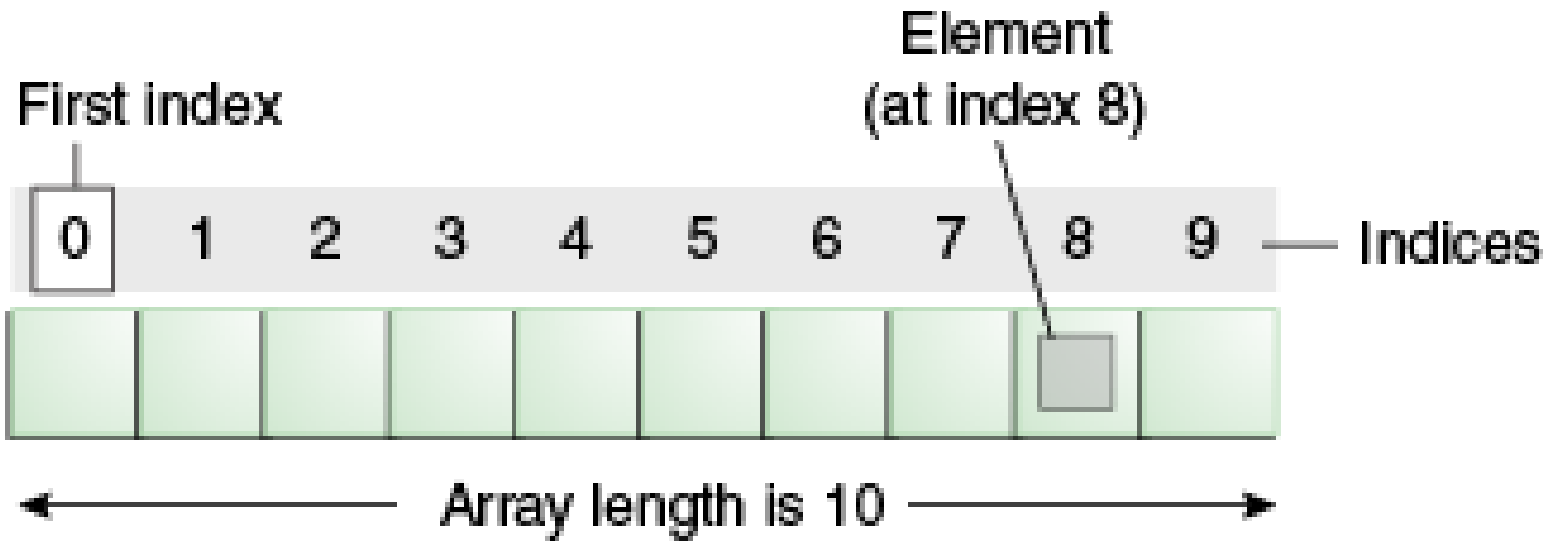
int x; 

A box called 'x' in which we can put one integer

- We can:
  - change the value stored completely,
  - add one to it,
  - subtract one from it etc.

- However this box can hold only one value. Imagine a bigger box made up of sub-divisions or sections. Such a box is called an array and would look like:

Structure of a primitive array

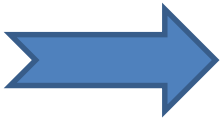


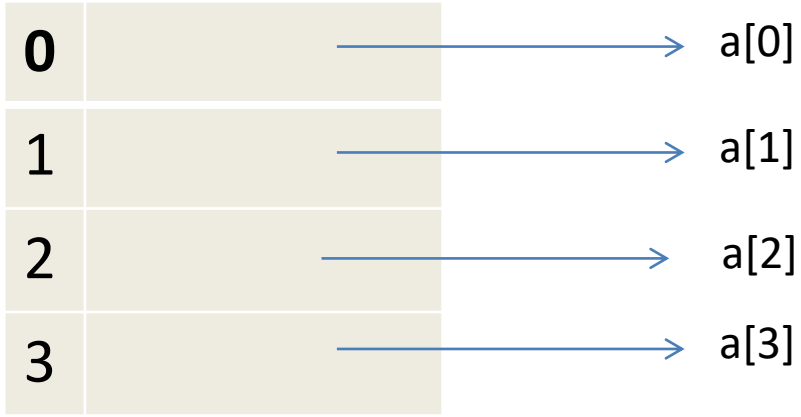
http://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Declaring a primitive array

- This is a box made up of four sub-divisions called 0, 1, 2 and 3.
- **NOTE :THE FIRST POSITION IS 0.**

```
int[]  a;
a = new int[4];
```

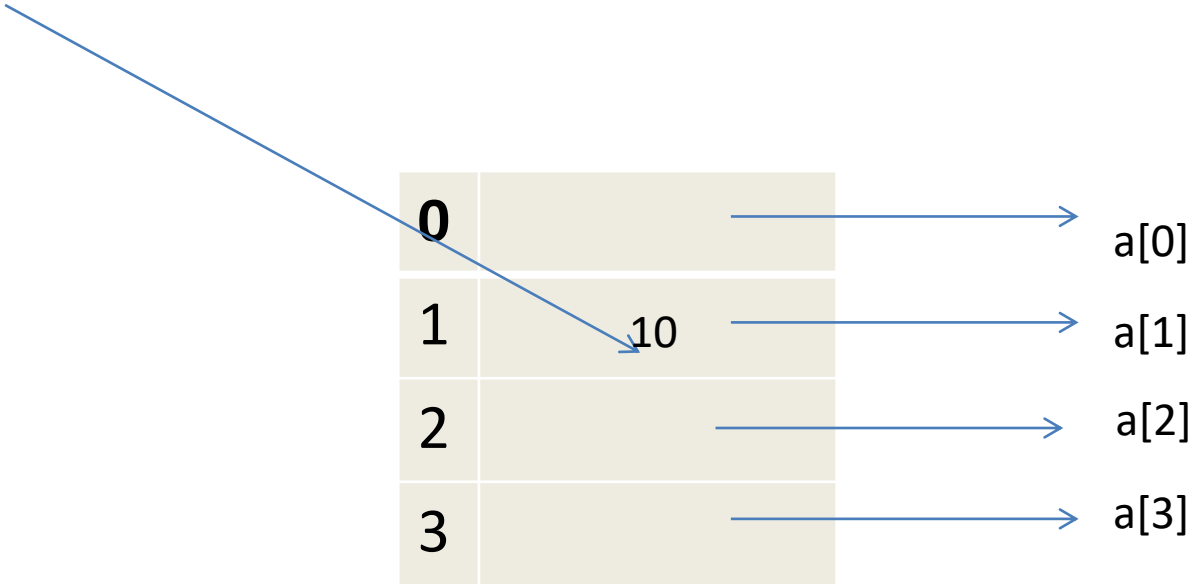




Accessing elements of an array

- You can access any element separately, e.g.

  a[1] = 10;



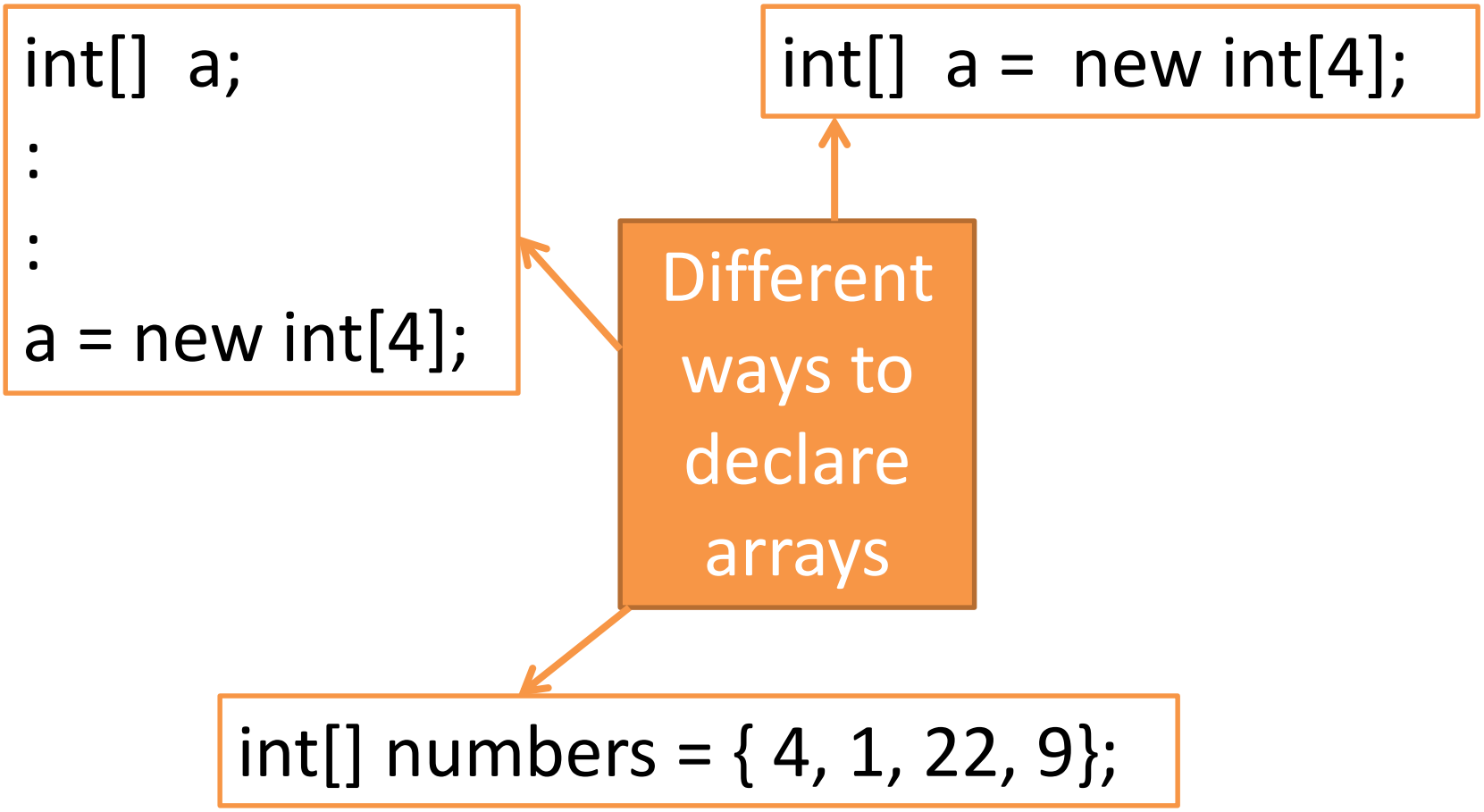
Rules for primitive Arrays

1. When you declare an array of a specific type, each sub-section (element) of the array is of the same declared type.

2. The size of the array, i.e. how many sections (elements) in the array is denoted by the number in the square bracket in the following statement:

```
int[] a = new int[4];
```

Declaring primitive arrays



Declaring an Array using literals

declaration and initialisation at the same time

```
private int[] numbers = { 3, 15, 4, 5 };



System.out.println(numbers[1]);
```

NOTE: literals can only be used when declaring an array.

Standard array use

```
private int[] hourCounts;
private String[] names;
private Person[] crowd;

...

hourCounts = new int[24];

...

hourcounts[i] = 0;
hourcounts[i]++;
System.out.println(hourcounts[i]);
```

Topic List

- Primitive arrays
  - Why do we need them?
  - What are they?
  - Using a primitive array.
  - Recap: for and while loops.
  - Arrays and counter-controlled loops.
  - Arrays and sentinel-based loop.
  - Arrays and flag-based loops.
  - Do you have to use all elements in the array?

Returning to our method that reads in and sums 5 numbers (typed in from the keyboard)…

and converting it to use primitive arrays…

Using arrays to remember numbers

```
public void askForFiveNumbersUsingArrays1()
{
    int numbers[] = new int[5];
    int sum = 0;
    for (int i = 0; i<5; i++)
    {
        System.out.print("Please enter a number: ");
        numbers[i] = scanner.nextInt();
    }

    for (int i = 0; i<5; i++)
    {
        sum += numbers[i];
    }
    System.out.println("The sum of the values you typed in is: " + sum);
}
```

- Using arrays
- Separate loop to add up the numbers

Using arrays to remember numbers

We could, of course sum the values immediately as they come in

```
public void askForFiveNumbersUsingArrays2()
{
    int numbers[] = new int[5];
    int sum = 0;
    for (int i = 0; i<5; i++)
    {
        System.out.print("Please enter a number: ");
        numbers[i] = scanner.nextInt();
        sum += numbers[i];
    }
    System.out.println("The sum of the values you typed in is: " + sum);
}
```

Using arrays with any size

If we wanted to change how many numbers we want to add…

```
public void askForFiveNumbersUsingArrays3(int size)
{
    int numbers[] = new int[size];
    int sum = 0;
    for (int i = 0; i < numbers.length; i++)
    {
        System.out.print("Please enter a number: ");
        numbers[i] = scanner.nextInt();
        sum += numbers[i];
    }
    System.out.println("The sum of the values you typed in is: " + sum);
}
```

Using arrays with any size

Asking the user how many numbers they want to enter…

```
public void askForFiveNumbersUsingArrays4()
{
    System.out.print("How many numbers to you want to enter: ");
    int size = scanner.nextInt();
    int numbers[] = new int[size];

    int sum = 0;
    for (int i = 0; i < numbers.length; i++)
    {
        System.out.print("Please enter a number: ");
        numbers[i] = scanner.nextInt();
        sum += numbers[i];
    }
    System.out.println("The sum of the values you typed in is: " + sum);
}
```

Array length

```
private int[] numbers = { 3, 15, 4, 5 };



int n = numbers.length;
```

No brackets! Length is NOT a method.

```
for(int i = 0; i < numbers.length; i++)
{
    System.out.println(i + ": " + numbers[i]);
}
```

What types can be stored in arrays?

- An array can store any type of data i.e.:
  - Object Types or
  - Primitive Types

```
int a[] = new int[10];

String words = new String[30];

Circle circles[] = new Circle[20];
```

Topic List

- Primitive arrays
  - Why do we need them?
  - What are they?
  - Using a primitive array.
  - Recap: for and while loops.
  - Arrays and counter-controlled loops.
  - Arrays and sentinel-based loop.
  - Arrays and flag-based loops.
  - Do you have to use all elements in the array?

Recap: for loops

- There are two variations of the for loop:
  - *for*

    The for loop is often used to iterate a fixed number of times.

    Often used with a variable that changes a fixed amount on each iteration.
  - *for-each*

    We used the for-each loop to iterate over a flexible-sized collection e.g. ArrayList.

Recap: for and while loop

General form of a for loop

```
for(initialization; condition; post-body action) {
    statements to be repeated
}
```

Equivalent in while-loop form

```
initialization;
while(condition) {
    statements to be repeated
    post-body action
}
```

Topic List

- Primitive arrays
  - Why do we need them?
  - What are they?
  - Using a primitive array.
  - Recap: for and while loops.
  - Arrays and counter-controlled loops.
  - Arrays and sentinel-based loop.
  - Arrays and flag-based loops.
  - Do you have to use all elements in the array?

Counter-controlled loops (arrays)

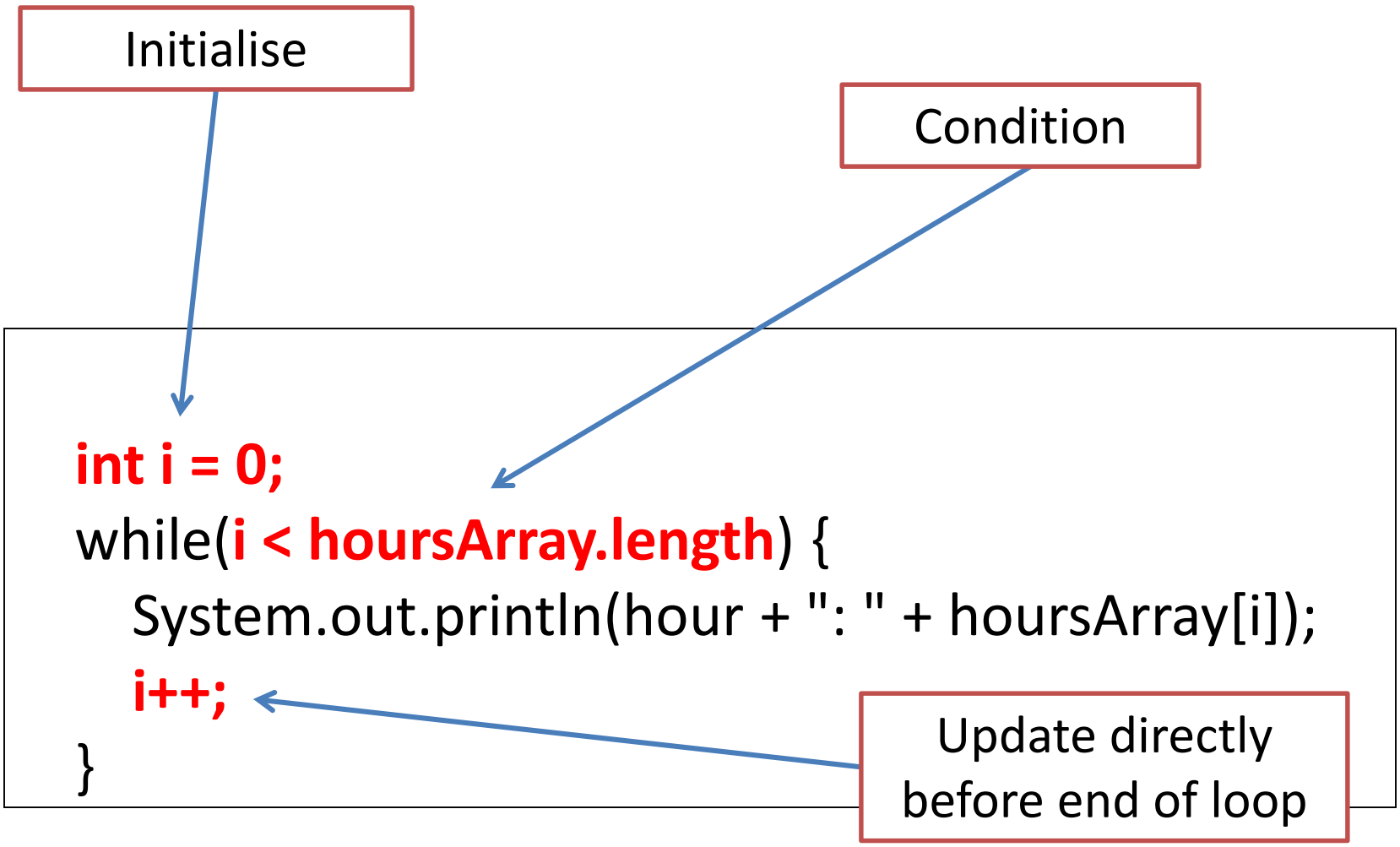
for loop version

```
for(int i = 0; i < hoursArray.length; i++) {
    System.out.println(hour + ": " + hoursArray[i]);
}
```

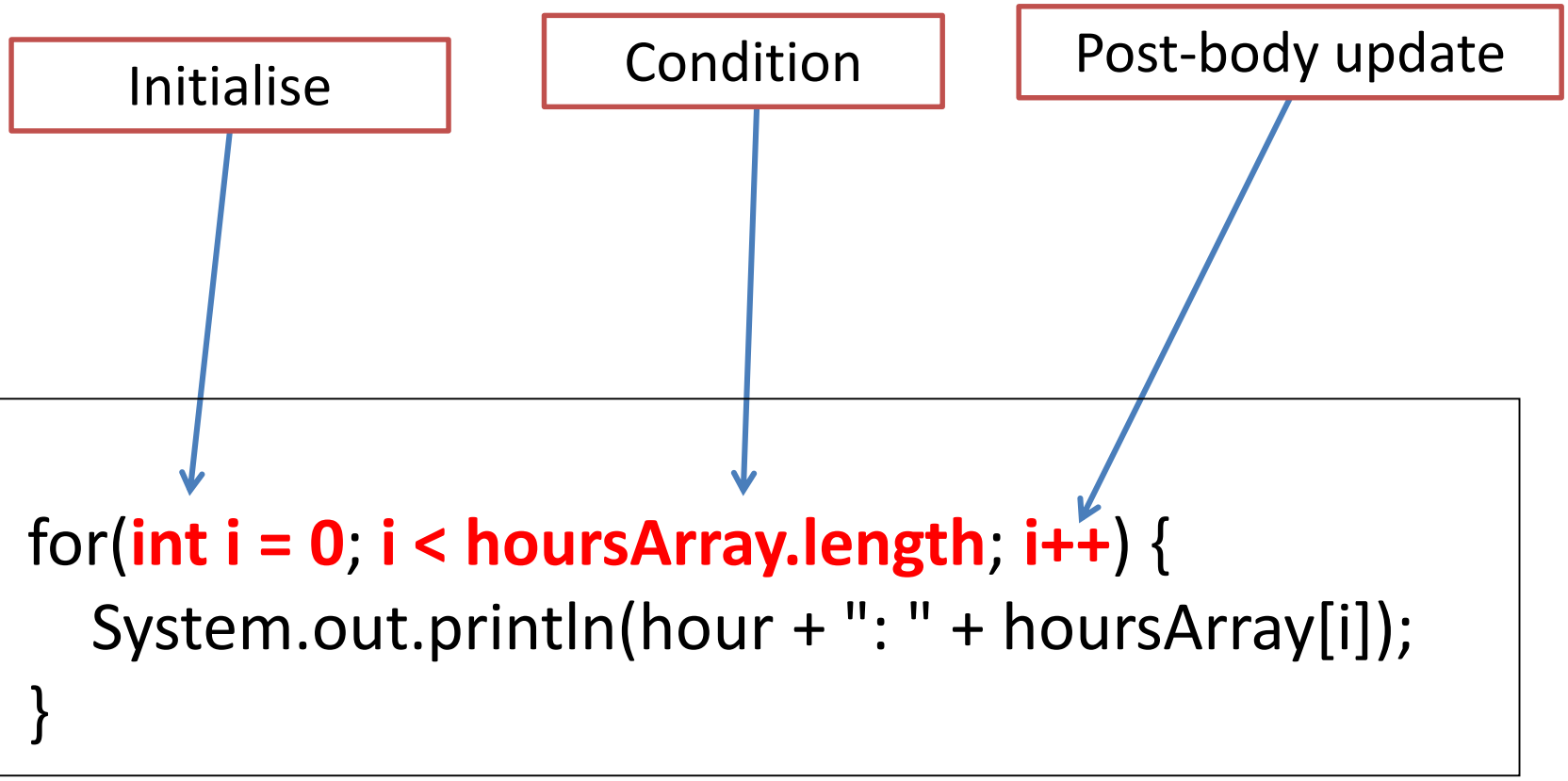
while loop version

```
int i = 0;
while(i < hoursArray.length) {
    System.out.println(hour + ": " + hoursArray[i]);
    i++;
}
```

Loop Control Variable (while)



Loop Control Variable (for)



Some for loop practice

Given an array of numbers, print out all the numbers in the array, using a for loop.

```
int[] numbers = { 4, 1, 22, 9, 14, 3, 9};

for ...
```

Your output should look like this →

```
Number 1: 4
Number 2: 1
Number 3: 22
Number 4: 9
Number 5: 14
Number 6: 3
Number 7: 9
```

Solution

```
public void forLoopPractice1()
{
    int[] numbers = { 4, 1, 22, 9, 14, 3, 9};

    for (int i = 0; i < numbers.length; i++)
    {
        System.out.println("Number " + (i+1) + ": " + numbers[i]);
    }
}
```

Your output should look like this →

```
Number 1: 4
Number 2: 1
Number 3: 22
Number 4: 9
Number 5: 14
Number 6: 3
Number 7: 9
```

More for loop practice

Fill an array with the Fibonacci sequence.

0 1 1 2 3 5 8 13 21 34 ...

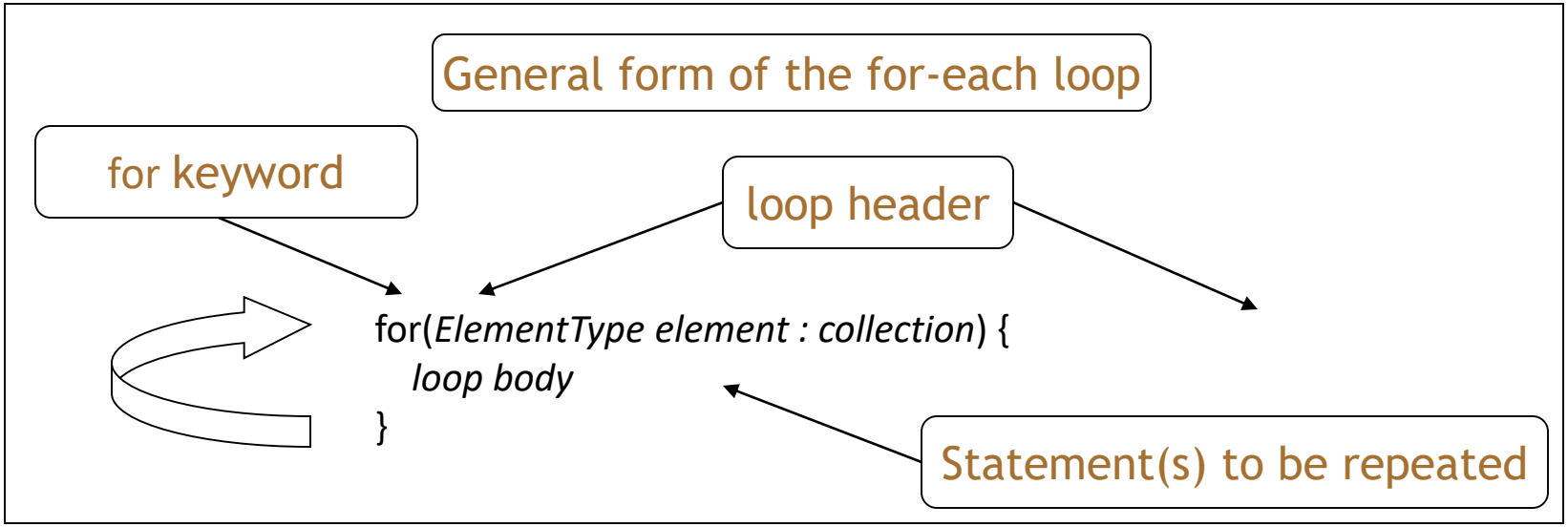
```
int[] fib = new int[40];

fib[0] = 0;
fib[1] = 1;

for ...
```

Solution

```
public void forLoopPractice2()
{
    int[] fib = new int[40];

    fib[0] = 0;
    fib[1] = 1;

    for (int i = 2; i < fib.length; i++)
    {
        fib[i] = fib[i-1] + fib[i-2];
    }

    for (int i = 0; i < fib.length; i++)
    {
        System.out.println("Number " + (i+1) + ": " + fib[i]);
    }
}
```

```
Number 1: 0
Number 2: 1
Number 3: 1
Number 4: 2
Number 5: 3
Number 6: 5
Number 7: 8
Number 8: 13
Number 9: 21
Number 10: 34
Number 11: 55
Number 12: 89
Number 13: 144
Number 14: 233
Number 15: 377
....
Number 39: 39088169
Number 40: 63245986
```

Recap: for-each loop pseudo-code



Pseudo-code expression of the actions of a for-each loop

For each *element* in *collection*, do the things in the *loop body*.

Some for each loop practice

- Given an array of numbers, print out all the numbers in the array, using a for each loop.

```
int[] numbers = { 4, 1, 22, 9, 14, 3, 9};

for ...
```

Your output should look like this →

```
Number: 4
Number: 1
Number: 22
Number: 9
Number: 14
Number: 3
Number: 9
```

Solution

```
Number: 4
Number: 1
Number: 22
Number: 9
Number: 14
Number: 3
Number: 9
```

```java
public void forLoopPractice3()
{
    int[] numbers = { 4, 1, 22, 9, 14, 3, 9};

    for (int currentNumber : numbers)
    {
        System.out.println("Number: " + currentNumber);
    }
}
```

Some for each loop practice

- Continuing to use a for each loop, refactor the code on the previous slide to include a count for each number printed.

```
int[] numbers = { 4, 1, 22, 9, 14, 3, 9};

for ...
```

Your output should now look like this →

Number **1**: 4
Number **2**: 1
Number **3**: 22
Number **4**: 9
Number **5**: 14
Number **6**: 3
Number **7**: 9

Solution

```
Number 1: 4
Number 2: 1
Number 3: 22
Number 4: 9
Number 5: 14
Number 6: 3
Number 7: 9
```

```java
public void forLoopPractice4()
{
    int[] numbers = { 4, 1, 22, 9, 14, 3, 9};

    int i = 1;
    for (int currentNumber : numbers)
    {
        System.out.println("Number " + (i) + ": " + currentNumber);
        i++;
    }
}
```

Topic List

- Primitive arrays
  - Why do we need them?
  - What are they?
  - Using a primitive array.
  - Recap: for and while loops.
  - Arrays and counter-controlled loops.
  - Arrays and sentinel-based loop.
  - Arrays and flag-based loops.
  - Do you have to use all elements in the array?

while Loops

- How do we control a while loop when we don't know how many inputs we will have?

  e.g. 'average of ages of people in the room', if you don't know how many are in the room.

```
initialization;
while(boolean condition) {
    statements to be repeated
    post-body action
}
```

while-loop

Sentinel-based loops

- We will signal the end of input with a special value i.e. a sentinel value.

  e.g. the code on the next slide continually asks the user to enter a person's age. When the user enters a value of -1, the loops ends and the total of all the ages is printed to the console.

Solution

```
public void sentinelBasedWhileLoop()
{
    int sum = 0;

    System.out.print("Please enter an age (-1 ends input): ");
    int ageInput = scanner.nextInt();

    while (ageInput != -1)
    {
        sum += ageInput;
        System.out.print("Please enter an age (-1 ends input): ");
        ageInput = scanner.nextInt();
    }

    System.out.println("The sum of the ages in the room is: " + sum);
}
```

Initialise

LCV Condition

Update LCV directly before end of loop

Sentinel-based loops - structure

- Concept of Loop Control Variable is vital here.
- The loop continuation is solely based on the input, so the variable containing the information is the Loop Control Variable.
- Initialise the Loop Control Variable before entry into the loop.
- Remember to ‘update the Loop Control Variable’ just before the end of the loop.

Topic List

- Primitive arrays
  - Why do we need them?
  - What are they?
  - Using a primitive array.
  - Recap: for and while loops.
  - Arrays and counter-controlled loops.
  - Arrays and sentinel-based loop.
  - Arrays and flag-based loops.
  - Do you have to use all elements in the array?

while Loops

- How do we control a while loop when we are looking for a specific property in a collection?

  e.g. test an array of numbers to see if any numbers are odd.

```
initialization;
while(boolean condition) {
    statements to be repeated
    post-body action
}
```

while-loop

Flag-Based Loops

- These are used when you want to examine a collection of data to check for a property.

- Once this property has been established, it cannot be ‘unestablished’:

  – ‘Once the flag is raised, it cannot by taken down’

Code to check ‘any numbers odd’

```
public void flagBasedWhileLoop()
{
    int[] numbers = { 4, 1, 22, 9, 14, 3, 9};
    boolean oddNumberInArray = false;

    for (int currentNumber : numbers)
    {
        if (currentNumber % 2 == 1)
        {
            oddNumberInArray = true;
        }
    }

    if (oddNumberInArray == true)
    {
        System.out.println("There is at least one odd number in the array");
    }
    else
    {
        System.out.println("There are NO odd numbers in the array");
    }
}
```

Slightly better code..

```
public void flagBasedWhileLoopBetter()
 {
     int[] numbers = { 4, 1, 22, 9, 14, 3, 9};
     boolean oddNumberInArray = false;

     for (int currentNumber : numbers)
     {
         if (currentNumber % 2 == 1)
         {
             oddNumberInArray = true;
         }
     }

     if (oddNumberInArray)
     {
         System.out.println("There is at least one odd number in the array");
     }
     else
     {
         System.out.println("There are NO odd numbers in the array");
     }
 }
```

Use of boolean variable in condition

What about having a

flag-based loop with a

boolean return type?

```java
public boolean flagBasedWhileLoopReturn()
{
    int[] numbers = { 4, 1, 22, 9, 14, 3, 9};

    for (int currentNumber : numbers)
    {
        if (currentNumber % 2 == 1)
        {
            return true;
        }
    }

    return false;
}
```

Code with boolean return type

Calling the method and handling the returned boolean

```java
if (flagBasedWhileLoopReturn()){
    System.out.println("There is at least one odd number in the array");
}
else{
    System.out.println("There are NO odd numbers in the array");
}
```

Topic List

- Primitive arrays
  - Why do we need them?
  - What are they?
  - Using a primitive array.
  - Recap: for and while loops.
  - Arrays and counter-controlled loops.
  - Arrays and sentinel-based loop.
  - Arrays and flag-based loops.
  - Do you have to use all elements in the array?

Do we have to use all elements in the array?

- No. We may not know how many elements of the array will actually be used e.g.
  - We wish to store an average mark for each of the 50 students in a particular class → create an array of 50 elements.
  - However, not all students might have sat their assessments; perhaps only 45 did → only 45 of the elements will be populated with an average mark.

Do we have to use all elements in the array?

- When not all elements in an array are populated, we need to:
  - have another variable (e.g. int size) which contains the number of elements of the array is actually used.
  - ensure size is used when processing the array e.g.

```
for (int i= 0; i < size; i++)
```

- For now, though, we assume that all elements of the array are populated and therefore ready to be processed.

Questions?

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/