MANCHESTER 1824
The University of Manchester

Introduction to Java

BIOL60201
Programming skills

Jean-Marc Schwartz
Faculty of Life Sciences

jean-marc.schwartz@manchester.ac.uk

Outline

1. Introduction to Java

2. Control flow and arrays
   i. Conditions
   ii. Loops
   iii. User input
   iv. Arrays

Forming conditions

- *if* statements:

```
if(condition)     //  If this is true
  statement;      //  then do this

if(condition1)          //  If condition1 is true
  statement1;           //  then do this;
else if(condition2)     //  else if condition2 is true
  statement2;           //  then do this;
else                    //  otherwise
  statement3;           //  do this
```

Relational operators

- `if(age < 33)` // is less than
- `if(age <= 33)` // is less than or equal to
- `if(age > 33)` // is greater than
- `if(age >= 33)` // is greater than or equal to
- `if(age == 33)` // is equal to
- `if(age != 33)` // is not equal to

- Note: the equal operator (==) is different from the assignment operator (=).

Logical operators

- `a && b` // a AND b
- `a || b` // a OR b
- `!a` // NOT a
- Example:

```
if(age >= 30 && age < 40) {
   // If the age is between 30 and 39, do something;
}
else if(age < 30) {
   // if the age is below 30, do something else;
}
else {
   // otherwise (if the age is 40 or more) do something else.
}
```

Switch statement

- The *switch* statement can be used instead of *if/else* when the condition consists of comparing an expression to integers or characters.
- Example: print the day of the week.

```
int day;

switch(day) {
  case 1: System.out.println("Monday");
          break;
  case 2: System.out.println("Tuesday");
          break;
  case 3: System.out.println("Wednesday");
          break;
  // etc
}
```

Conditional operator

- The conditional operator evaluates an expression and assigns it to one of two possibilities depending on it being true or false.

```
(condition ? expression1 : expression2)
```

  If the condition is true, expression 1 is applied; otherwise, expression 2 is applied.

- Example: compute the absolute value.

```
abs = (a >= 0 ? a : -a);
```

for loops

- `for(initial statement; loop condition; loop update) {`
  `// Loop body`
  `}`

  1. Execute the initial statement.
  2. Test the loop condition.
  3. If true execute the loop body, otherwise exit the loop.
  4. Execute the loop update.
  5. Repeat from 2.

for loops

- Example:

```
int i;
for(i=1; i<=10; i++) {
  System.out.print(3*i + " ");
}
```

Result:

```
3 6 9 12 15 18 21 24 27 30
```

Nested *for* loops

- Example:

```
int i, j;
for(i=1; i<=5; i++) {
  for(j=1; j<=5; j++) {
    System.out.print(i*j + " ");
  }
  System.out.println();
}
```

Result:

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25
```

while loops

- ```
  while(loop condition) {
     // Loop body
  }
  ```

  1. Test the loop condition.
  2. If true execute the loop body, otherwise exit the loop.
  3. Repeat from 1.

- Note: if the loop condition is false from beginning, the loop body is not executed at all.
  ```

while loops

- Example:

```
int i=0;

while(i < 10) {
   System.out.print(3*i + " ");
   i++;
}
```

Result:

```
0 3 6 9 12 15 18 21 24 27
```

do/while loops

- ```
  do {
     //  Loop body
  }
  while(loop condition);
  ```

  1. Execute the loop body
  2. Test the loop condition.
  3. If true repeat from 1, otherwise exit the loop.

- Note: the loop body is always executed at least once, because the test is at the end.
  ```

do/while loops

- Example:

```
int i=0;

do {
   System.out.print(3*i + " ");
   i++;
}
while(i < 10);
```

Result:

```
0 3 6 9 12 15 18 21 24 27
```

Type-safe user input

- Example:

```
import java.util.Scanner;

public class AgeInput {

  public static void main(String[] args) {

    Scanner scan = new Scanner(System.in);
    System.out.print("Enter your age: ");

    while(!scan.hasNextInt()) {
      scan.nextLine();
      System.out.print("Please enter an integer: ");
    }

    int age = scan.nextInt();
    System.out.println("Your age is " + age);
    scan.close();
  }
}
```

Validating user input

- Example:

```
import java.util.Scanner;

public class MonthInput {

  public static void main(String[] args) {

    int month;
    Scanner scan = new Scanner(System.in);

    do {
      System.out.print("Enter a month between 1 and 12: ");
      month = scan.nextInt();
    } while(month<1 || month>12);

    System.out.println("Your chosen month is: " + month);
    scan.close();
  }
}
```

Arrays

- An array is a sequence of variables of the same type, enabling us to store multiple values under a single variable name.
- Each item is called an element of the array.
- Each element is identified and accessed by a subscript or index.
- The size and data type must be specified upon initialisation.
- For an array of length SIZE, indices range from zero to SIZE-1.

Declaring and instantiating arrays

- Declaration:
```
datatype[] arrayName;
```

- Instantiation:
```
arrayName = new datatype[size];
```

- Examples:

```
double[] temperatures;
temperatures = new double[31];

final int QUESTIONS = 10;
boolean answers[] = new boolean[QUESTIONS];

int values[] = {1,2,3,5,7,11};
```

Accessing array elements

- Examples:

```
int size = 4;
double[] bills = new double[size];

bills[0] = 45.30;
bills[1] = 12.89;
bills[2] = 64.99;
bills[3] = 21.00;

double total = 0;
for(int i=0; i<size; i++) {
  total += bills[i];
  }

System.out.println("Total spending: " + total);
```

Multidimensional arrays

- It is possible to define arrays with more than one dimension.

- Example:

```
int table[][] = new int[4][10];
   //  Defines a matrix of 4 rows and 10 columns

for(int i=0; i<4; i++) {
   for(int j=0; j<10; j++) {
      table[i][j] = i*10 + j;
   }
}
```

Multidimensional arrays

- Accessing the content of a two-dimensional array:

```
for(int i=0; i<table.length(); i++) {

  for(int j=0; j<table[i].length(); j++) {

    System.out.print("Row " + i + ", " +
                     "Column " + j + ": " +
                     table[i][j] + "\n");
  }
}
```

Generating random numbers

- The *Random* class provides methods to generate random numbers.
- The *nextInt(number)* method returns a random integer ranging from 0 up to, but not including, *number*.
- Example:

```
import java.util.Random;

public class DieNumbers {

  public static void main(String[] args) {

    Random random = new Random();
    int die = random.nextInt(6) + 1;
    System.out.println("The die roll is: " + die);
  }
}
```