3. Controlling Program Flow

Prof. Dr. Charles Wüthrich
B.Sc. Francesco Andreussi,
CoGVis/MMC, Faculty of Media
Bauhaus-University Weimar

Introduction

- Until now, our programs ran from their head down until the end of the program was reached
- Java offers ways of controlling when parts of the code are executed, and when not.
- Decisions are based on *comparison operators*, which base on *boolean operations.*

Comparison operators

- Take operands of ONE type and produce an output of type `boolean`:

Oper.	Meaning	TRUE	FALSE
==	IsEqual	6 == 6	6 == 8
!=	IsNotEqual	6 != 8	8 != 8
<	IsSmaller	37 < 64	15 < 15
<=	IsSmallerOrEqual	12 <= 12	12 <= 7
>	IsBigger	45 > 27	45 > 51
>=	IsBiggerOrEqual	31 >= 31	31 >= 60

Comparison operators

- One can combine comparisons into multiple comparisons:
  - Example: `anint` is an integer variable

- The comparison

```
((2 <= anint) && (20>=anint))
```

  results in ?.....

Comparison operators

- One can combine comparisons into multiple comparisons:
  - Example: `anint` is an integer variable

- The comparison

```
((2 <= anint) && (20>=anint))
```

  results in
  - TRUE if `anint` is between 2 and 20
  - FALSE if not.

IF statements

```
public class SwapAB {
   public static void main(String[] args) {
      int a,b,c;

      // assign a to 1st command line paramenter
      a = Integer.parseInt(args[0]);
      // assign b to 2nd command line paramenter
      b = Integer.parseInt(args[1]);
      if(b<a){
         c=a;
         a=b;
         b=c;
      } // Swap a and b if necessary
      System.out.printf("a: %d, b: %d%n",a,b);
   } // main
} // SwapAB
```

If statements

- Executing we obtain:

```
> java SwapAB 200 22
```

  which executes and outputs:

```
a: 22, b: 200
```

- Anybody can tell me why we need c?

If-Then-Else

- This statement provides an alternate path of execution when the IF condition is FALSE:

```
if(SomeCondition){
   DoSomething;
}
else{
   DoSomethingElse;
}
```

- Anybody can tell me what happens here?

Else IF

- The IF-ELSE can be expanded as here:

```
public class IncomeStatement {
   public static void main(String[] args) {
      int income;

      // assign income to 1st command line paramenter
      income = Integer.parseInt(args[0]);
      if(income<14000){
        System.out.println("Your income is a low income");
        } // Swap a and b if necessary
      else if(income<40000){
        System.out.println("Your income is average");
      }
      else{
        System.out.println("You swim in gold");
      }
   } // main
} // IncomeStatement
```

While

- *Loop*: A group if instructions is repeated over and over again until a certain result is achieved.

- First construct of this type:

```
while(SomeCondition){
   DoSomething;
}
```

- Remember the recipe?

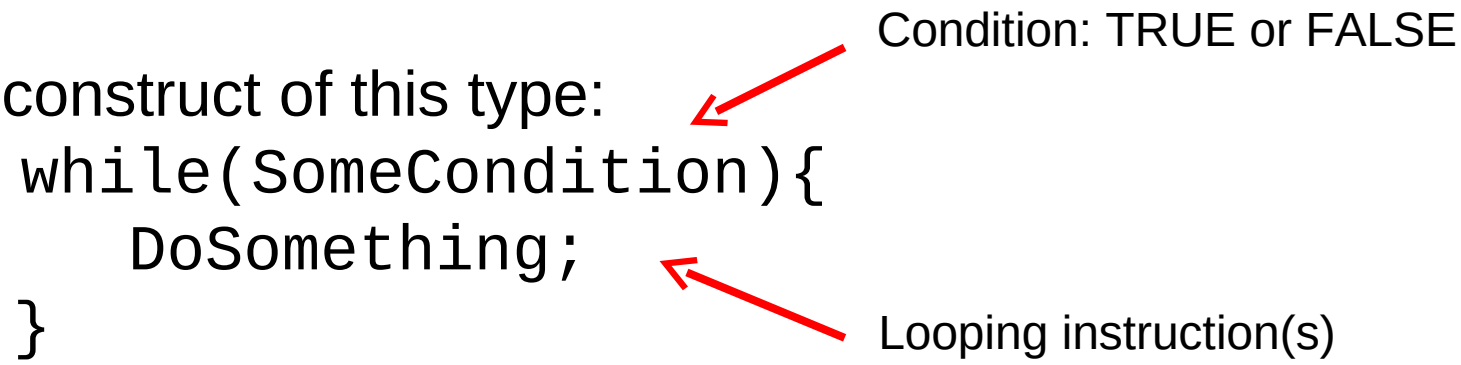
```
Cook at low flame until brown
```

- Exactly same construct

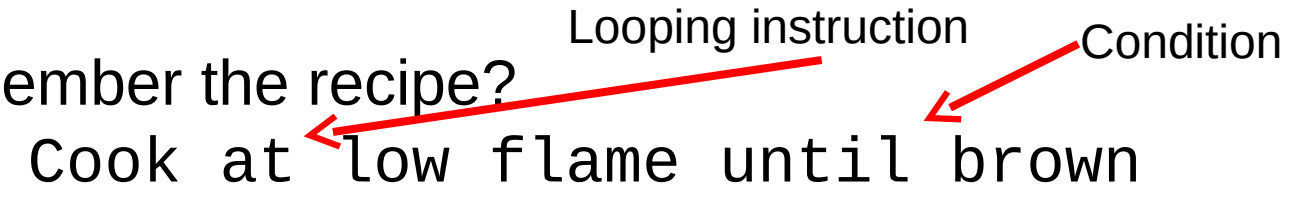
While

- *Loop*: A group if instructions is repeated over and over again until a certain result is achieved.

- First construct of this type:



- Remember the recipe?



While

- Example:
- We are riding a bike at the speed of 12 km/h
- We apply the brake until we come to a standstill:
  - We pull the brake
  - Decrease the speed by 1 km/h
  - Keep doing this until our speed is 0

While

- Example:

```
public class DecreaseSpeed {
   public static void main(String[] args) {
      int vel=12; // initial speed in km/h

      while(vel >= 0){
         vel=vel-1;
         System.out.println("Current speed "+vel+" km/h");
      }
      System.out.println("I have come to a standstill");
   } // main
} // DecreaseSpeed
```

While

- Execution of the program in the CLI

```
> java DecreaseSpeed
Current speed 11 km/h
Current speed 10 km/h
Current speed 9 km/h
Current speed 8 km/h
Current speed 7 km/h
Current speed 6 km/h
Current speed 5 km/h
Current speed 4 km/h
Current speed 3 km/h
Current speed 2 km/h
Current speed 1 km/h
Current speed 0 km/h
I have come to a standstill
>
```

For

- The FOR instruction is another way to do loops:
  we use a control variable (which has been predeclared as the `int i;`)

```
for(i=12; i>0; i=i-1){
   DoSomething;
}
```

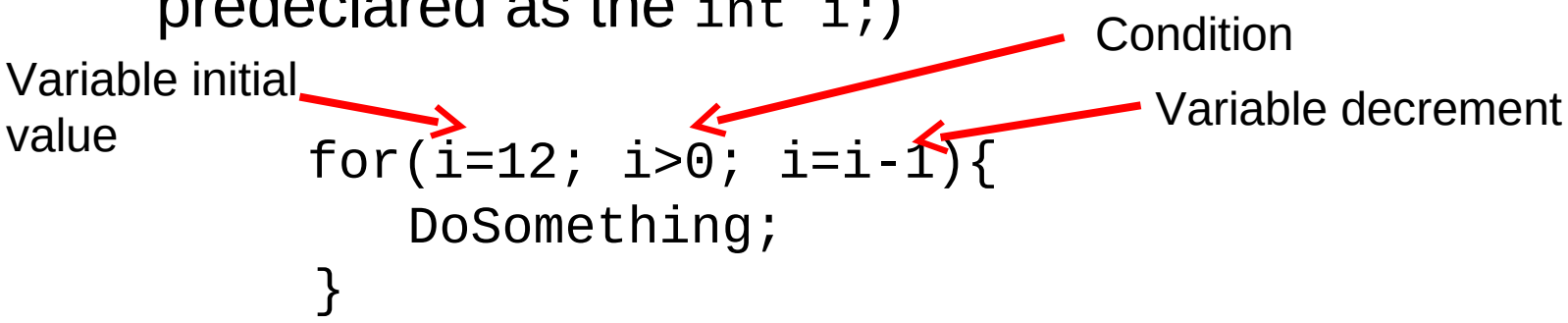
- Notice: in Java
  - `i=i-1` ⇔ `i--`
  - `i=i+1` ⇔ `i++`
- So `for(i=12; i>0; i=i--){`

```
   DoSomething;
}
```

  does the same

For

- The FOR instruction is another way to do loops: we use a control variable (which has been predeclared as the `int i;`)



- Let's rewrite the previous WHILE loop as a FOR

For

- Example:

```
public class DecreaseSpeed2 {
   public static void main(String[] args) {
      int vel; // speed

      for(vel=12; vel>0; vel=val-1){
        System.out.println("Current speed "+vel+" km/h");
      }
      System.out.println("I have come to a standstill");
   } // main
} // DecreaseSpeed2
```

For

- Execution from the CLI:

```
> java DecreaseSpeed2
Current speed 12 km/h
Current speed 11 km/h
Current speed 10 km/h
Current speed 9 km/h
Current speed 8 km/h
Current speed 7 km/h
Current speed 6 km/h
Current speed 5 km/h
Current speed 4 km/h
Current speed 3 km/h
Current speed 2 km/h
Current speed 1 km/h
I have come to a standstill
>
```

Break

- The *break* instruction interrupts the execution of a loop.
- It is widely used in loops where the computer needs input from a user
- Example: continuously ask the user to input a character until the user presses “q”

Break

- Example: continuously ask the user to input a character until the user presses “q”

```
public class ReadUserInputQuit {
   public static void main(String[] args) {
      char input='0'; // contains what the user typed
      String inputstring;

      while(input != 'q'){
        System.out.println("Type a character:");
        inputstring = System.console().readLine();
        input = inputstring.charAt(0);
        System.out.println("You typed "+input);
      }
      System.out.println("q is the exit character");
   } // main
} // ReadUserInputQuit
```

Break

- Example: continuously ask the user to input a character until the user presses “q”

```
public class ReadUserInputQuit {
   public static void main(String[] args) {
      char input='0'; // contains what the user typed
      String inputstring;

      while(input != 'q'){
        System.out.println("Type a character:");
        inputstring = System.console().readLine();
        input = inputstring.charAt(0);
        System.out.println("You typed "+input);
      }
      System.out.println("q is the exit character");
   } // main
} // ReadUserInputQuit
```

Put Command line into string

Extract 1st character of string

Break

- Example: CLI execution

```
> java ReadUserInputQuit
Type a character:
s
You typed s
Type a character:
z
You typed z
Type a character:
y
You typed y
Type a character:
q
You typed q
q is the exit character
>
```

Break

- Doing the same with a `break`:

```
public class ReadUserInputQuit2 {
   public static void main(String[] args) {
      char input='0'; // contains what the user typed
      String inputstring;

      while(true){
        System.out.println("Type a character:");
        inputstring = System.console().readLine();
        input = inputstring.charAt(0);
        System.out.println("You typed "+input);
        if(input == 'q'){
           break;
        }
      }
      System.out.println("q is the exit character");
   } // main
} // ReadUserInputQuit2
```

Breaks the loop

Break

- Doing the same with a `break`:

```
public class ReadUserInputQuit2 {
   public static void main(String[] args) {
      char input='0'; // contains what the user typed
      String inputstring;

      while(true){
        System.out.println("Type a character:");
        inputstring = System.console().readLine();
        input = inputstring.charAt(0);
        System.out.println("You typed "+input);
        if(input == 'q'){
           break;
        }
      }
      System.out.println("q is the exit character");
   } // main
} // ReadUserInputQuit2
```

Infinite loop!

Break

- Example: CLI execution

```
> java ReadUserInputQuit2
Type a character:
s
You typed s
Type a character:
z
You typed z
Type a character:
y
You typed y
Type a character:
q
You typed q
q is the exit character
>
```

Switch

- The switch statement is used to simplify multiple if-then-else statements
- It works only on `char`, `byte`, `short` and `int`.
- For example:

```
switch(i){
    case 0: DoSomething;
            break;
    case 1: DoSomethingElse;
            break;
    case 2: DoYetSometingElse;
            break;
}
```

Switch

- The program on the right
  - asks the user to type in the numeric value of a weekday
  - Converts it to corresponding string
  - Prints it

```
public class DayOfTheWeek {
  public static void main(String[] args) {
    int input=-1; // contains what the user typed
    String inputstring;
    String day_of_week="Sunday"; // starting value

    System.out.println("Type the number of the day [0-6]:");
    inputstring = System.console().readLine();
    input = Integer.parseInt(inputstring);

    switch(input){
      case 0: day_of_week="Sunday";
              break;
      case 1: day_of_week="Monday";
              break;
      case 2: day_of_week="Tuesday";
              break;
      case 3: day_of_week="Wednesday";
              break;
      case 4: day_of_week="Thursday";
              break;
      case 5: day_of_week="Friday";
              break;
      case 6: day_of_week="Saturday";
              break;
      }
      System.out.println("Day "+input+" is "+day_of_week);
  } // main
} // DayOfTheWeek
```

Switch

- Executing the program from the CLI one obtains:

```
> java DayOfTheWeek
Type the number of the day [0-6]:
5
Day 5 is Friday
>
```

- Notice: all `case` statements have been skipped except 5

Do-While

- Similar to WHILE, but the condition is checked after the first execution of the loop, and not before

```
do{
    SomeInstructionStatements};
} while(ConditionFullfilled);
```

End

+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++