Control flow in Java

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 2

Control flow

- Program layout
  - Statements end with semi-colon
  - Blocks of statements delimited by braces

Control flow

- Program layout
  - Statements end with semi-colon
  - Blocks of statements delimited by braces
- Conditional execution
  - `if (condition) { ... } else { ... }`

Control flow

- Program layout
  - Statements end with semi-colon
  - Blocks of statements delimited by braces
- Conditional execution
  - `if (condition) { ... } else { ... }`
- Conditional loops
  - `while (condition) { ... }`
  - `do { ... } while (condition)`

Control flow

- Program layout
  - Statements end with semi-colon
  - Blocks of statements delimited by braces
- Conditional execution
  - `if (condition) { ... } else { ... }`
- Conditional loops
  - `while (condition) { ... }`
  - `do { ... } while (condition)`
- Iteration
  - Two kinds of `for`

Control flow

- Program layout
  - Statements end with semi-colon
  - Blocks of statements delimited by braces
- Conditional execution
  - `if (condition) { ... } else { ... }`
- Conditional loops
  - `while (condition) { ... }`
  - `do { ... } while (condition)`
- Iteration
  - Two kinds of `for`
- Multiway branching – `switch`

Conditional execution

- `if (c) {...} else {...}`
  - `else` is optional
  - Condition must be in parentheses
  - If body is a single statement, braces are not needed
- No `elif`, à la Python
  - Indentation is not forced
  - Just align `else if`
  - Nested `if` is a single statement, no separate braces required
- No surprises
- Aside: no `def` for function definition

```
public class MyClass {

  ...

  public static int sign(int v) {
    if (v < 0) {
      return(-1);
    } else if (v > 0) {
      return(1);
    } else {
      return(0);
    }
  }

}
```

Conditional loops

- `while (c) {...}`
  - Condition must be in parentheses
  - If body is a single statement, braces are not needed

```
public class MyClass {

  ...

  public static int sumupto(int n) {
    int sum = 0;

    while (n > 0){
      sum += n;
      n--;
    }

    return(sum);
  }

}
```

Conditional loops

- `while (c) {...}`
  - Condition must be in parentheses
  - If body is a single statement, braces are not needed
- `do {...} while (c)`
  - Condition is checked at the end of the loop
  - At least one iteration

```
public class MyClass {

  ...

  public static int sumupto(int n) {
    int sum = 0;
    int i = 0;

    do {
      sum += i;
      i++;
    } while (i <= n);

    return(sum);
  }
```

Conditional loops

- `while (c) {...}`
  - Condition must be in parentheses
  - If body is a single statement, braces are not needed
- `do {...} while (c)`
  - Condition is checked at the end of the loop
  - At least one iteration
  - Useful for interactive user input

```
do {
  read input;
} while (input-condition);
```

```
public class MyClass {

  ...

  public static int sumupto(int n) {
    int sum = 0;
    int i = 0;

    do {
      sum += i;
      i++;
    } while (i <= n);

    return(sum);
  }
```

Iteration

- `for` loop is inherited from C
- `for (init; cond; upd) {...}`
  - `init` is initialization
  - `cond` is terminating condition
  - `upd` is update

Iteration

- `for` loop is inherited from C
- `for (init; cond; upd) {...}`
  - `init` is initialization
  - `cond` is terminating condition
  - `upd` is update
- Intended use is
  `for(i = 0; i < n; i++){...}`

```
public class MyClass {

  ...

  public static int sumarray(int[] a) {
    int sum = 0;
    int n = a.length;
    int i;

    for (i = 0; i < n; i++){
      sum += a[i];
    }

    return(sum);
  }
```

Iteration

- `for` loop is inherited from C
- `for (init; cond; upd) {...}`
  - `init` is initialization
  - `cond` is terminating condition
  - `upd` is update
- Intended use is
  `for(i = 0; i < n; i++){...}`
- Completely equivalent to

```
i = 0;
while (i < n) {
  i++;
}
```

```
public class MyClass {

  ...

  public static int sumarray(int[] a) {
    int sum = 0;
    int n = a.length;
    int i;

    for (i = 0; i < n; i++){
      sum += a[i];
    }

    return(sum);
  }
```

Iteration

- Intended use is
  `for(i = 0; i < n; i++){...}`
- Completely equivalent to

```
i = 0;
while (i < n) {
  i++;
}
```

```
public class MyClass {

  ...

  public static int sumarray(int[] a) {
    int sum = 0;
    int n = a.length;
    int i;

    for (i = 0; i < n; i++){
      sum += a[i];
    }

    return(sum);
  }
```

Iteration

- Intended use is
  `for(i = 0; i < n; i++){...}`
- Completely equivalent to

```
i = 0;
while (i < n) {
  i++;
}
```

- However, not good style to write `for` instead of `while`

```
public class MyClass {

  ...

  public static int sumarray(int[] a) {
    int sum = 0;
    int n = a.length;
    int i;

    for (i = 0; i < n; i++){
      sum += a[i];
    }

    return(sum);
  }
```

Iteration

- Intended use is
  `for(i = 0; i < n; i++){...}`
- Completely equivalent to

```
i = 0;
while (i < n) {
  i++;
}
```

- However, not good style to write `for` instead of `while`
- Can define loop variable within loop
  - The scope of `i` is local to the loop
  - An instance of more general local scoping allowed in Java

```
public class MyClass {

  ...

  public static int sumarray(int[] a) {
    int sum = 0;
    int n = a.length;

    for (int i = 0; i < n; i++){
      sum += a[i];
    }

    return(sum);
  }

}
```

Iterating over elements directly

- Java later introduced a `for` in the style of Python

```
for x in l:
  do something with x
```

Iterating over elements directly

- Java later introduced a `for` in the style of Python

```
for x in l:
  do something with x
```

- Again `for`, different syntax

```
for (type x : a)
  do something with x;
}
```

```
public class MyClass {

  ...

  public static int sumarray(int[] a) {
    int sum = 0;
    int n = a.length;

    for (int v : a){
      sum += v;
    }

    return(sum);
  }
```

Iterating over elements directly

- Java later introduced a `for` in the style of Python

```
for x in l:
  do something with x
```

- Again `for`, different syntax

```
for (type x : a)
  do something with x;
}
```

- It appears that loop variable must be declared in local scope for this version of `for`

```
public class MyClass {

  ...

  public static int sumarray(int[] a) {
    int sum = 0;
    int n = a.length;

    for (int v : a){
      sum += v;
    }

    return(sum);
  }
```

Multiway branching

- `switch` selects between different options

```
public static void printsign(int v) {
  switch (v) {
    case -1: {
      System.out.println("Negative");
      break;
    }
    case 1: {
      System.out.println("Positive");
      break;
    }
    case 0: {
      System.out.println("Zero");
      break;
    }
  }
}
```

Multiway branching

- `switch` selects between different options
- Be careful, default is to "fall through" from one case to the next
  - Need to explicitly `break` out of switch
  - `break` available for loops as well
  - Check the Java documentation

```
public static void printsign(int v) {
  switch (v) {
    case -1: {
      System.out.println("Negative");
      break;
    }
    case 1: {
      System.out.println("Positive");
      break;
    }
    case 0: {
      System.out.println("Zero");
      break;
    }
  }
}
```

Multiway branching

- `switch` selects between different options
- Be careful, default is to "fall through" from one case to the next
  - Need to explicitly `break` out of switch
  - `break` available for loops as well
  - Check the Java documentation
- Options have to be constants
  - Cannot use conditional expressions

```
public static void printsign(int v) {
  switch (v) {
    case -1: {
      System.out.println("Negative");
      break;
    }
    case 1: {
      System.out.println("Positive");
      break;
    }
    case 0: {
      System.out.println("Zero");
      break;
    }
  }
}
```

Multiway branching

- `switch` selects between different options
- Be careful, default is to "fall through" from one case to the next
  - Need to explicitly `break` out of switch
  - `break` available for loops as well
  - Check the Java documentation
- Options have to be constants
  - Cannot use conditional expressions
- Aside: here return type is `void`
  - Non-`void` return type requires an appropriate `return` value

```
public static void printsign(int v) {
  switch (v) {
    case -1: {
      System.out.println("Negative");
      break;
    }
    case 1: {
      System.out.println("Positive");
      break;
    }
    case 0: {
      System.out.println("Zero");
      break;
    }
  }
}
```

Summary

- Program layout: semi-colons, braces
- Conditional execution: `if`, `else`
- Conditional loops: `while`, `do-while`
- Iteration: two kinds of `for`
  - Local declaration of loop variable
- Multiway branching: `switch`
  - `break` to avoid falling through