

Repetition structures

Repetition structures

Repetition structures allow a program to repeat executing sequences of instructions under certain conditions.

The most basic form of repetition structures is the **while** statement:

```
while (<condition>) {  
    <statements>  
}
```

while loops

The most basic form of repetition structures is the **while** statement:

```
while (<condition>) {  
    <statements>  
}
```

- <condition> is a *boolean* expression.
- <statements> can be any sequence of statements.

while loops

In the **while** statement

```
while (<condition>) {  
    <statements>  
}
```

<statements> is repeatedly executed *over and over again* as long as <condition> is **true**. The loop terminates only when <condition> becomes **false**.

while loops

In the **while** statement

```
while (<condition>) {  
    <statements>  
}
```

just like the **if** statement, if <condition> is **false** to start with, then <statements> is never executed.

Meaningful loops

```
while (<condition>) {  
    <statements>  
}
```

In order for this loop to be useful, the value of <condition> must become from **true** to **false** at some point while <statements> is repeatedly executed.

Meaningful loops

Example.

```
int i = 0
while (i < 10) {
    System.out.print(i + " ");
    i++;
}
```

This simply means that, as long as $i < 10$, print the value of i and increment i by 1.

Meaningful loops

Example.

```
int i = 0
while (i < 10) {
    System.out.print(i + " ");
    i++;
}
```

Output: 0 1 2 3 4 5 6 7 8 9

Meaningful loops

Example.

```
int i = 0
while (i < 10) {
    System.out.print(i + " ");
    i++;
}
```

`i++;` is the statement that changes the value of `i < 10` from **true** to **false**.

for loops

In Java, **for** statements have the form:

```
for (<initialization>; <condition>; <update>) {  
    <statements>  
}
```

- <initialization> is a statement to *execute once before* starting the loop.
- <condition> is a boolean expression.
- <update> is a statement to execute after *each iteration*.

for loops

```
for (<initialization>; <condition>; <update>) {  
    <statements>  
}
```

is equivalent to:

```
<initialization>;  
while (<condition>) {  
    <statements>  
    <update>;  
}
```

for loops

Example. A **for** loop to print numbers 0 through 99:

```
for (int n = 0; n < 100; n++)  
    System.out.println(n);
```

for loops

```
for (int n = 0; n < 100; n++)  
    System.out.println(n);
```

is equivalent to:

```
int n = 0;  
while (n < 100) {  
    System.out.println(n);  
    n++;  
}
```

do-while loops

do-while statements have the form:

```
do {  
    <statements>  
} while (<condition>) {
```

- `<condition>` is evaluated at the end.
- It is useful when we always need to **execute** `<statements>` at least once.

do-while loops

Example. This can be simplified using a **do-while** loop.

```
Scanner s = new Scanner(System.in);  
int sum = 0;  
System.out.print("Enter a number: ");  
int n = s.nextInt();  
while (n > 0) {  
    sum = sum + n;  
    System.out.print("Enter a number: ");  
    n = s.nextInt();  
}  
System.out.println("The sum of is " + sum + ".");
```

do-while loops

Example. This can be simplified using a **do-while** loop.

```
Scanner s = new Scanner(System.in);  
int sum = 0;  
int n = 0;  
do {  
    sum = sum + n;  
    System.out.print("Enter a number: ");  
    n = s.nextInt();  
} while (n > 0);  
System.out.println("The sum of is " + sum + ".");
```

Nested loops

Example. What does this do?

```
for (int i = 1; i <= 12; i++) {  
    for (int j = 1; j <= 12; j++)  
        System.out.print(i*j + " ");  
    System.out.println();  
}
```

It generates a multiplication table.

Nested loops

Example. What does this do?

```
for (int i = 1; i <= 12; i++) {  
    for (int j = 1; j <= 12; j++) {  
        if (i*j < 10) System.out.print("  ");  
        else if (i*j < 100) System.out.print(" ");  
        System.out.print(i*j + " ");  
    }  
    System.out.println();  
}
```

It generates a multiplication table.