# Iteration

## `while` Loops

- Executes a block of code repeatedly
- A condition controls how often the loop is executed

```
while (condition)
   statement;
```

- Most commonly, the statement is a block (set of statements delimited by `{ }`)

## Calculating the Growth of an Investment

- Invest $10,000, 5% interest, compounded annually

| Year | Balance |
|---|---|
| **0** | **$10,000** |
| **1** | **$10,500** |
| **2** | **$11,025** |
| **3** | **$11,576.25** |
| **4** | **$12,155.06** |
| **5** | **$12,762.82** |

## Calculating the Growth of an Investment

- When has the bank account reached a particular balance?

```
while (balance < targetBalance)
{
   year++;
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

## `while` Loop Flowchart

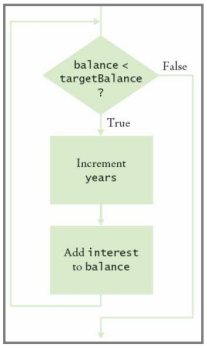


**Figure 1:**
**Flowchart of a `while` Loop**

## Syntax 7.1: The `while` Statement

```
while (condition)
   statement
```

**Example:**

```
while (balance < targetBalance)
{
   year++;
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

**Purpose:**
To repeatedly execute a statement as long as a condition is true

## Question

1. How many times is the statement in the loop executed?

```
while (false) statement;
```

2. What would happen if `rate` was set to `0` in the `main` method of the `InvestmentTester` program?

## Answers

1. 0 times
2. The `waitForBalance` method would never return due to an infinite loop

## Common Error: Infinite Loops

- 
```
int years = 0;
while (years < 20)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```
- 
```
int years = 20;
while (years > 0)
{
   years++; // Oops, should have been years--
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```
- Loops run forever–must kill program

## Common Error: Off-By-One Errors

```
int years = 0;
while (balance < 2 * initialBalance)
{
   years++;
   double interest = balance * rate / 100;
   balance = balance + interest;
}
System.out.println("The investment reached the target after "
   + years + " years.");
```

- Should `years` start at 0 or 1?
- Should the test be < or <=?

## Avoiding Off-by-One Error

- Look at a scenario with simple values:
  initial `balance`: $100
  interest `rate`: 50%
  after year 1, the `balance` is $150
  after year 2 it is $225, or over $200
  so the investment doubled after 2 years
  the loop executed two times, incrementing `years` each time
  *Therefore*: `years` must start at 0, not at 1.

## Avoiding Off-by-One Error

- interest rate: 100% after one year: `balance` is 2 * `initialBalance` loop should stop
  *Therefore:* must use <
- Think, don't compile and try at random

## do Loops

- Executes loop body at least once:

```
do
   statement
while (condition);
```

- Example: Validate input

```
double value;
do
{
   System.out.print("Please enter a positive number: ");
   value = in.nextDouble();
}
while (value <= 0);
```

## do Loops

- Alternative:

```
boolean done = false;
while (!done)
{
   System.out.print("Please enter a positive number: ");
   value = in.nextDouble();
   if (value > 0) done = true;
}
```

## do Loop Flowchart

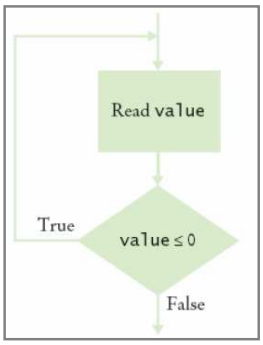


**Figure 2:**
**Flowchart of a do Loop**

## for Loops

- 

```
for (initialization; condition; update)
   statement
```

Example:

```
for (int i = 1; i <= n; i++)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

## for Loops

- Equivalent to

```
initialization;
while (condition)
{ statement; update; }
```

- Other examples:

```
for (years = n; years > 0; years--) . . .
```

```
for (x = -10; x <= 10; x = x + 0.5) . . .
```

## Flowchart for for Loop

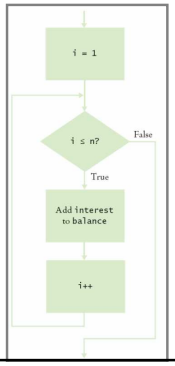


**Figure 4:**
**Flowchart of a for Loop**

## Syntax 7.2: The `for` Statement

```
for (initialization; condition; update)
   statement
```

**Example:**

```
for (int i = 1; i <= n; i++)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

**Purpose:**
To execute an initialization, then keep executing a statement and updating an expression while a condition is true

## Common Errors: Semicolons

- A semicolon that shouldn't be there

```
sum = 0;
for (i = 1; i <= 10; i++);
   sum = sum + i;
System.out.println(sum);
```

- A missing semicolon

```
for (years = 1; (balance = balance + balance *
   rate / 100) < targetBalance; years++)
System.out.println(years);
```

## Nested Loops

- Create triangle pattern

```
[]
[][]
[][][]
[][][][]
```

- Loop through rows

```
for (int i = 1; i <= n; i++)
{
   // make triangle row
}
```

## Nested Loops

- *Make triangle row* is another loop

```
for (int j = 1; j <= i; j++)
   r = r + "[]";
r = r + "\n";
```

- Put loops together → Nested loops

## Processing Sentinel Values

- Sentinel value: Can be used for indicating the end of a data set
- `0` or `-1` make poor sentinels; better use `Q`

```
System.out.print("Enter value, Q to quit: ");
String input = in.next();
if (input.equalsIgnoreCase("Q"))
   We are done
else
{
   double x = Double.parseDouble(input);
   . . .
}
```

## Loop and a half

- Sometimes termination condition of a loop can only be evaluated in the middle of the loop
- Then, introduce a boolean variable to control the loop:

```
boolean done = false;
while (!done)
{
   Print prompt String input = read input;
   if (end of input indicated)
      done = true;
   else
   {
      // Process input
   }
}
```

## Random Numbers and Simulations

- In a simulation, you repeatedly generate random numbers and use them to simulate an activity
- Random number generator

```
Random generator = new Random();
int n = generator.nextInt(a); // 0 <= n < a
double x = generator.nextDouble(); // 0 <= x < 1
```

- Throw die (random number between 1 and 6)

```
int d = 1 + generator.nextInt(6);
```