Announcements

- The wednesday, February 4th, lecture will be held at ENGMD 280 (McDonald Engineering 280)

Properties of conditionals

- In the following, C is any boolean expression, P, Q, R , S, and T are any list of statements.

```
P;
if (C) {
  Q;
  R;
}
else{
  Q;
  S;
}
T;
```

Properties of conditionals

is equivalent to

```
P;
Q;
if (C) {
  R;
}
else {
  S;
}
T;
```

if and only if the statements in Q do not modify the variables in C

Properties of conditionals

- Consider the following:

```
boolean high = false;
double altitude;
altitude = Keyboard.readDouble();
System.out.println(“Begin”);
if (altitude > 2000.0) {
  high = true;
  System.out.println(“It is high”);
}
else {
  high = true;
  System.out.println(“It is low”);
}
```

Properties of conditionals

- It is equivalent to:

```
boolean high = false;
double altitude;
altitude = Keyboard.readDouble();
System.out.println(‘‘Begin’’);
high = true;
if (altitude > 2000.0) {
  System.out.println(‘‘It is high’’);
}
else {
  System.out.println(‘‘It is low’’);
}
```

Properties of conditionals

- Consider the following:

```
double altitude;
altitude = Keyboard.readDouble();
System.out.println(“Begin”);
if (altitude > 2000.0) {
  altitude = altitude - 500.0;
  System.out.println(“It is high”);
}
else {
  altitude = altitude - 500.0;
  System.out.println(“It is low”);
}
```

Properties of conditionals

- It is *not* equivalent to:

```
double altitude;
altitude = Keyboard.readDouble();
System.out.println(“Begin”);
altitude = altitude - 500.0;
if (altitude > 2000.0) {
  System.out.println(“It is high”);
}
else {
  System.out.println(“It is low”);
}
```

Conditionals

```
int x;
boolean b;
//...
if (b) {
  x=3;
}
else {
  x=4;
}
```

is equivalent to

```
x=4;
if (b) {
  x=3;
}
```

Conditionals

```
int x,y;
boolean b;
//...
if (b) {
  x=3;
}
else {
  y=4;
}
```

is *not* equivalent to

```
y=4;
if (b) {
  x=3;
}
```

Problem solving

- Clear statement of the problem
- Analysis (of the problem)
- Design
- Implementation
- Testing / Verification
- Maintenance

Analysis

- Goal: to obtain a precise understanding the problem
- Things to do in analysis:
  - Determine inputs and outputs
  - Determine general and specific requirements
  - Make or obtain precise definitions of concepts involved
  - Determine the relevant information to the problem
  - Determine the relationship between different elements or pieces of information of the problem
  - Make explicit any relevant assumptions

Design

- Goal: to obtain an algorithm or set of algorithms which solves the problem correctly, satisfying all of the problem's requirements

- An algorithm is an (abstract) procedure which describes the solution to a problem

- Develop an algorithm using different techniques:
  - Decision diagrams
  - Incremental design
  - Divide and conquer
  - Dynamic programming
  - etc.

- Develop data-structures required by the algorithm(s)

- Design a general structure or organization of the set of algorithms

Implementation

- Goal: to realize an algorithm or set of algorithms into a computer program, using a programming language

- Implementation depends on the particular programming language being used.

- Concretise the general organization by dividing the system into modules

- In Object-Oriented programming:
  - Describe information and data structures as classes
  - Translate algorithms into methods

Testing

- Goal: to gain confidence in that the program solves the problem adequately and without errors
- Testing involves:
  - Identify key features to be tested
  - Defining test cases which cover all significan aspects
  - Performing the tests (possibly in an automatic way)
- A program which has been tested satisfactorily is not guarranteed to be correct (because it is impossible to always cover all possible cases.)
- To be certain of absolute correctness, the design and the implementation must be mathematically *proven* to be correct. This is called *verification*. This is different than testing.

Maintenance

- Goal: to make appropriate modifications to a program if required

- Maintenance might be required when
  - the program generates errors (compile-time or run-time)
  - the specification of the problem changes
  - the program should be improved (e.g. speed, better user-interface, etc.)

- Maintenance might require changes at:
  - the implementation level (debugging)
  - the design level
  - the analysis level

Conditionals

- Problem: compute the taxes to be paid by a person depending on the person's single/married status, if the person is filing jointly with his/her spouse, and the taxable income of that person, according to the following:
  - A single person earning no more than $21,450, or a married person filing jointly and earning less than $35,800, pays 15% of all income.
  - A single person earning between $21,450 and $51,900, pays a base amount of $3,217.50 plus 28% of the income amount over $21,450.
  - A married person filing jointly, earning between $35,800 and $86,500, pays a base amount of $5,370.00 plus 28% of the income amount over $35,800.
  - A single person earning more than $51,900 pays a base amount of $11,743.50 plus 31% of the income amount over $51,900.
  - A married person filing jointly, earning more than $86,500 pays a base amount of $19,566.00 plus 31% of the income amount over $86,500.

Analysis

- Inputs:
  - Whereas married and filing jointly or filing as single
  - Taxable income
- Output: tax
- Other relevant information:
  - Tax brackets
  - Base amount payable for each tax bracket
  - Cutoff for each tax bracket
  - Rates for each tax bracket
- Assumptions: tax brackets, base amounts, cutoffs and rates are fixed
- Assumptions: taxable income is greater or equal to $0

Analysis

- Relationships:
  - If filing as single:

If the taxable income is over	but not over	the tax is	of the amount over
$0	$21,450	15%	$0
$21,450	$51,900	$3,217.50+28%	$21,450
$51,900		$11,743.50+31%	$51,900

  - If filing jointly:

If the taxable income is over	but not over	the tax is	of the amount over
$0	$35,800	15%	$0
$35,800	$86,500	$5,370.00+28%	$35,800
$86,500		$19,566.00+31%	$86,500

Analysis

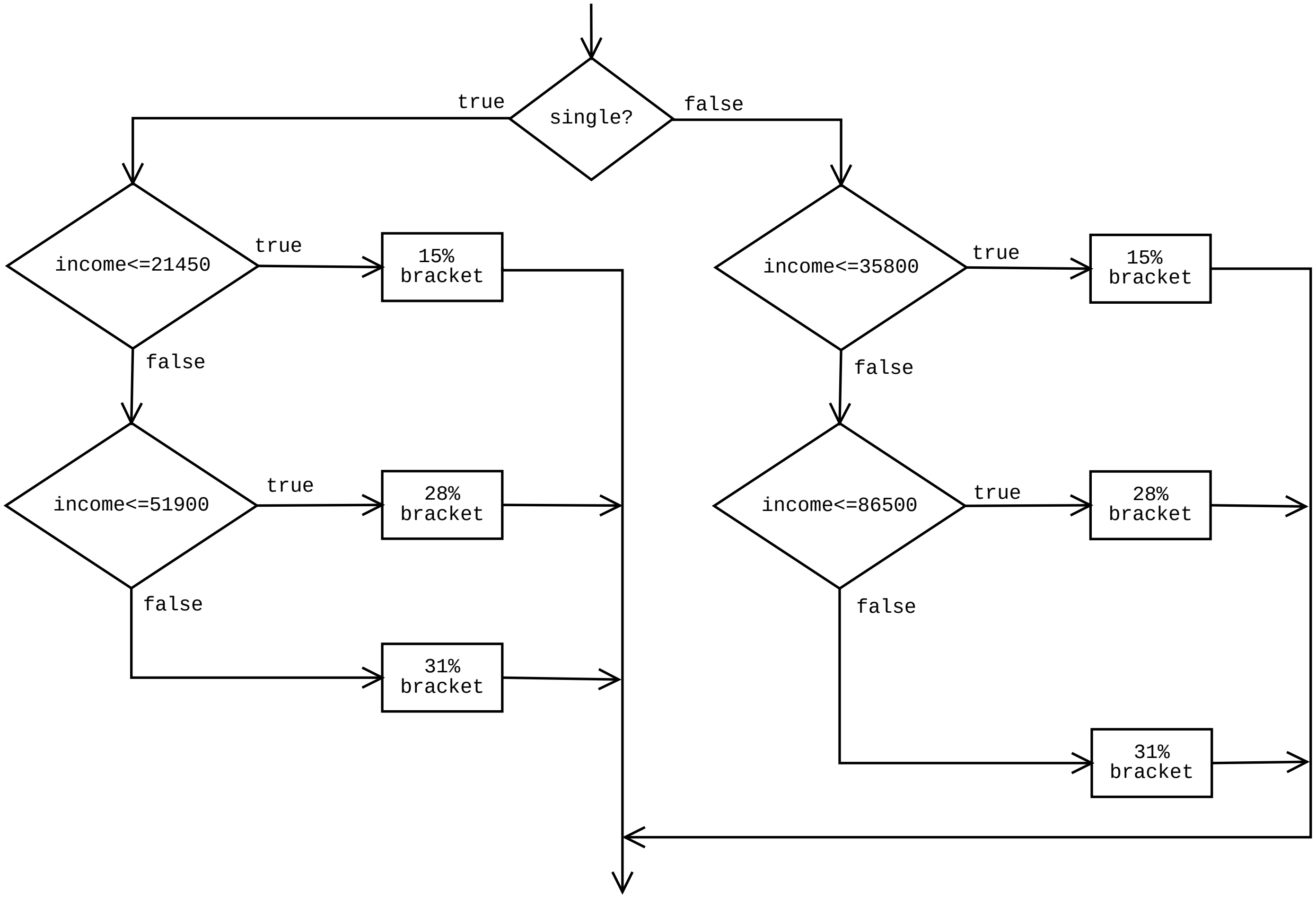
- The tax is computed (by definition) according to the following equality

$$tax = base + rate \times (income - cutoff)$$

- For example:
  - If a single person earns $30,000, then the base is $3,217.50, the rate is 28% and the cutoff is $21,450, so the tax will be

$$tax = 3217.50 + 0.28 \times (30000.0 - 21450.0)$$

Design


single?
true
false
income<=21450
true
15% bracket
false
income<=51900
true
28% bracket
false
31% bracket
income<=35800
true
15% bracket
false
income<=86500
true
28% bracket
false
31% bracket

Implementation

```
import cs1.Keyboard;
public class TaxCalculator {
  public static void main(String[] args) {
    double income;
    boolean single_status;
    double tax;
    String single;

    System.out.print(“Enter your taxable income: ”
    income = Keyboard.readDouble;
    System.out.print(“Are you filing as single? (y
    single = Keyboard.readString();
    single = single.toLowerCase();
    if (single.equals(“yes”))
      single_status = true;
    else single_status = false;

    if (single_status) {
      if (income <= 21450.00) {
        tax = income * 0.15;
      }
      else if (income <= 51900.00) {
        tax = 3217.50 + 0.28 * (income - 21450.00)
      }
      else {
        tax = 11743.50 + 0.31 * (income - 51900.00
      }
    }
    else { // filing as married
      if (income <= 35800.00) {
        tax = income * 0.15;
      }
      else if (income <= 86500.00) {
        tax = 5370.00 + 0.28 * (income - 35800.00)
      }
      else {
        tax = 19566.00 + 0.31 * (income - 86500.00
      }
    }

    System.out.println(“The tax payable is ”+tax);

  } // End of main method
} // End of TaxCalculator class
```

Implementation

```
import cs1.Keyboard;
public class TaxCalculator {
  public static void main(String[] args) {
    double income;
    boolean single_status;
    double tax, base, rate, cutoff;
    String single;

    System.out.print(“Enter your taxable income: ”
    income = Keyboard.readDouble;
    System.out.print(“Are you filing as single? (y
    single = Keyboard.readString();
    single = single.toLowerCase();
    if (single.equals(“yes”))
      single_status = true;
    else single_status = false;

    if (single_status) {
      if (income <= 21450.00) {
        base = 0.00;
        rate = 0.15;
```

```
          cutoff = 0.00;
        }
        else if (income <= 51900.00) {
          base = 3217.50;
          rate = 0.28;
          cutoff = 21450.00;
        }
        else {
          base = 11743.50;
          rate = 0.31;
          cutoff = 51900.00;
        }
      }
      else { // filing as married
        if (income <= 35800.00) {
          base = 0.00;
          rate = 0.15;
          cutoff = 0.00;
        }
        else if (income <= 86500.00) {
          base = 5370.00;
          rate = 0.28;
          cutoff = 35800.00;
      }
      else {
        base = 19566.00;
        rate = 0.31;
        cutoff = 86500.00;
      }
    }
    tax = base + rate * (income - cutoff);

    System.out.println(“The tax payable is ”+tax);

  } // End of main method
} // End of TaxCalculator class
```

Constants

- To enforce that a variable cannot change we declare it as a constant:

  ```
  final type variable = expression;
  ```

- The variable must be initialised

  ```
  final double PI = 3.1415;
  PI = 2 * PI; // Error
  ```

- A variable declared as final is a constant and cannot ocurr on the left-hand side of an assignment statement

- It is common practice (but not mandatory) to name constants in all capitalized letters.

Implementation

```
import cs1.Keyboard;
public class TaxCalculator {
  public static void main(String[] args) {
    double income;
    boolean single_status;
    double tax, base, rate, cutoff;
    String single;

    final double SINGLE_CUTOFF_1 = 21450.00;
    final double SINGLE_CUTOFF_2 = 51900.00;
    final double MARRIED_CUTOFF_1 = 35800.00;
    final double MARRIED_CUTOFF_2 = 86500.00;
    final double SINGLE_BASE_1 = 3217.50;
    final double SINGLE_BASE_2 = 11743.50;
    final double MARRIED_BASE_1 = 5370.00;
    final double MARRIED_BASE_2 = 19566.00;
    final double RATE_1 = 0.15;
    final double RATE_2 = 0.28;
    final double RATE_3 = 0.31;
```

```
System.out.print(“Enter your taxable income: ”
income = Keyboard.readDouble;
System.out.print(“Are you filing as single? (y
single = Keyboard.readString();
single = single.toLowerCase();
if (single.equals(“yes”))
  single_status = true;
else single_status = false;

if (single_status) {
  if (income <= SINGLE_CUTOFF_1) {
    base = 0.00;
    rate = RATE_1;
    cutoff = 0.00;
  }
  else if (income <= SINGLE_CUTOFF_2) {
    base = SINGLE_BASE_1;
    rate = RATE_2;
    cutoff = SINGLE_CUTOFF_1;
  }
  else {
    base = SINGLE_BASE_2;
    rate = RATE_3;
    cutoff = SINGLE_CUTOFF_2;
  }
}
else { // filing as married
  if (income <= MARRIED_CUTOFF_1) {
    base = 0.00;
    rate = RATE_1;
    cutoff = 0.00;
  }
  else if (income <= MARRIED_CUTOFF_2) {
    base = MARRIED_BASE_1;
    rate = RATE_2;
    cutoff = MARRIED_CUTOFF_1;
  }
  else {
    base = MARRIED_BASE_2;
    rate = RATE_3;
    cutoff = MARRIED_CUTOFF_2;
  }
}
tax = base + rate * (income - cutoff);
```

```
   System.out.println(“The tax payable is ”+tax);

  } // End of main method
 } // End of TaxCalculator class
```

Abstraction

- Abstraction:

  "disassociated from any specific instance" - Webster's dictionary

- To abstract is to make something independent of particular cases

- Variables give us a basic mechanism for abstraction:
  - A concrete definition:

    $$tax = 3217.50 + 0.28 \times (income - 21450.0)$$

  - An abstract definition:

    $$tax = base + rate \times (income - cutoff)$$

- In software, abstraction facilitates reusability and makes it easier to maintain.

The random method

- The method

  ```
  static double random()
  ```

  from the Math class returns a random number between 0 and 1 (including 0 but excluding 1)

- It can be used for giving random integers in any interval by means of casting

  ```
  int coin;
  coin = (int)(Math.random() * 2);

  int die;
  die = (int)(Math.random() * 6 + 1);
  ```

Large conditionals

```
int die;
die = (int)(6 * Math.random() + 1);

if (die == 1)
  System.out.println(“Excellent”);
else
  if (die == 2)
    System.out.println(“Good”);
  else
    if (die == 3)
      System.out.println(“OK”);
    else
      if (die == 4)
        System.out.println(“Ah...”);
      else
        if (die == 5)
          System.out.println(“Bad”);
        else
          if (die == 6)
            System.out.println(“Terrible”);
```

Large conditionals

```
int die;
die = (int)(6 * Math.random() + 1);

if (die == 1)
  System.out.println(“Excellent”);
else if (die == 2)
  System.out.println(“Good”);
else if (die == 3)
  System.out.println(“OK”);
else if (die == 4)
  System.out.println(“Ah...”);
else if (die == 5)
  System.out.println(“Bad”);
else if (die == 6)
  System.out.println(“Terrible”);
```

The switch statement

```
int die;
die = (int)(6 * Math.random() + 1);
switch (die) {
  case 1:
    System.out.println(“Excellent”);
    break;
  case 2:
    System.out.println(“Good”);
    break;
  case 3:
    System.out.println(“OK”);
    break;
  case 4:
    System.out.println(“Ah...”);
    break;
  case 5:
    System.out.println(“Bad”);
    break;
  case 6:
    System.out.println(“Terrible”);
    break;
}
```

Large conditionals

```
int die;
die = (int)(6 * Math.random() + 1);

if (die == 1)
  System.out.println(“Excellent”);
else if (die == 2)
  System.out.println(“Good”);
else if (die == 3)
  System.out.println(“OK”);
else if (die == 4)
  System.out.println(“Ah...”);
else if (die == 5)
  System.out.println(“Bad”);
else
  System.out.println(“Terrible”);
```

The switch statement

```
int die;
die = (int)(6 * Math.random() + 1);
switch (die) {
  case 1:
    System.out.println(“Excellent”);
    break;
  case 2:
    System.out.println(“Good”);
    break;
  case 3:
    System.out.println(“OK”);
    break;
  case 4:
    System.out.println(“Ah...”);
    break;
  case 5:
    System.out.println(“Bad”);
    break;
  default:
    System.out.println(“Terrible”);
    break;
}
```

The switch statement

- Just another form of conditional

```
switch (integer_or_character_expression) {
  case integer_or_character_expression_1:
    list_of_statements_1;
    break;
  case integer_or_character_expression_2:
    list_of_statements_2;
    break;
  case integer_or_character_expression_3:
    list_of_statements_3;
    break;
   ...
  default:
    list_of_statements_n;
}
```

The switch statement

- Semantics:

1. Evaluate the condition,
   (a) compare it with each case
   (b) if a case matches, the corresponding list of statements is executed
      i. if there is a break statement, the switch stops and computation continues directly after the switch.
      ii. if there is no break statement in the list, execution continues with the next case

The switch statement

```
int die;
die = (int)(6 * Math.random() + 1);
switch (die) {
  case 1:
    System.out.println(“Excellent”);
    break;
  case 2:
    System.out.println(“Good”);
  case 3:
    System.out.println(“OK”);
  case 4:
    System.out.println(“Ah...”);
    break;
  case 5:
    System.out.println(“Bad”);
    break;
  default:
    System.out.println(“Terrible”);
    break;
}
```

The switch statement

- If the break statement is included,

```
switch (C) {
  case E1:
    S1;
    break;
  case E2;
    S2;
    break;
  case E3;
    S3;
    break;
  ...
  default:
    Sn;
}
```

is equivalent to

The switch statement

```
if (C == E1) S1;
else if (C == E2) S2;
else if (C == E3) S3;
...
else Sn;
```

Switch conditions

- An integer expression is an arithmetic expression of type int, short, long or byte, e.g.

```
3
5+3*-2
x*(7/2)        // (if x is of type int)
(int)‘A’
s.length()     // (if s is a String)
```

- The expression `(int)’A’` has as value the ASCII or Unicode number for the character `’A’`

Character expressions

- A character expression is an expression of type char

```
‘a’
‘B’
‘8’
‘ ’
‘d’ + 2
(char)65
s.charAt(3)          // if s is a String
```

- The expression `‘d’ + 2` has as value the character `’f’`

- The expression `(char)65` has as value the character corresponding to the ASCII or Unicode number 65 (`‘a’`)

- Character expressions can be used in relational expressions (Their ASCII or Unicode value is compared):

```
‘m’ <= ’p’
‘D’ > ‘A’
‘a’ < ‘A’
```

Character expressions

```
String sentence;
char c;
boolean letter = false, digit = false;

sentence = Keyboard.readString();
c = sentence.charAt( sentence.length() - 1 );

if (‘A’ <= c && c <= ‘Z’ || ‘a’ <= c && c <= ‘z’)
  letter = true;
else if (‘0’ <= c && c <= ‘9’)
  digit = true;
```

Character expressions

```
String sentence;
char c;

sentence = Keyboard.readString();
c = sentence.charAt( sentence.length() - 1 );

if (‘A’ <= c && c <= ‘Z’) {
  c = (char)(c + (‘a’ - ‘A’));
  // c is a lower case letter
}
```

Character expressions

```
String sentence;
char c;

sentence = Keyboard.readString();
c = sentence.charAt( sentence.length() - 1 );

if (‘a’ <= c && c <= ‘z’) {
  c = (char)(c + (‘A’ - ‘a’));
  // c is an upper case letter
}
```

Switch conditions

```
String name;
name = Keyboard.readString();

switch( name.charAt(3) - 2 ) {
  case ‘e’:
    System.out.println(“Helloooo”);
    break;
  case ‘h’:
    System.out.println(“Noooo”);
    break;
  case ‘z’:
    System.out.println(“OK”);
}
```

Character expressions

```
String sentence;
char c;
boolean vowel;

sentence = Keyboard.readString();
sentence = sentence.toLowerCase();
c = sentence.charAt( sentence.length() - 1 );

switch (c) {
  case ‘a’:
  case ‘e’:
  case ‘i’:
  case ‘o’:
  case ‘u’:
    vowel = true;
    break;
  default:
    vowel = false;
}
```

Statements

- Variable declaration

  ```
  type variable;
  ```

- Assignment

  ```
  variable = expression;
  ```

- Method invocation

  ```
  objectreference.methodname(parameters);
  ```

  or

  ```
  classname.methodname(parameters);
  ```

- Conditional

  ```
  if (condition) block;
  ```

  or

  ```
  if (condition) block1; else block2;
  ```

- Loop

Loops

- The loop is a statement used to describe a task which is *repetitive*
- For example: print the first 100 odd integers

```
System.out.println(1);
System.out.println(3);
System.out.println(5);
System.out.println(7);
System.out.println(9);
System.out.println(11);
System.out.println(13);
//...
```

- What if we want to print the first 1000 odd numbers?
- What if the user is supposed to give the program the number of odd numbers?

Loops

- The basic loop statement:

```
while (boolean_expression) {
  list_of_statements;
}
```

- Semantics: the execution of a while loop proceeds as follows:

1. The boolean expression is evaluated

   (a) If it is false,

      i. the loop stops

      ii. and computation proceeds directly after the loop

   (b) If it is true,

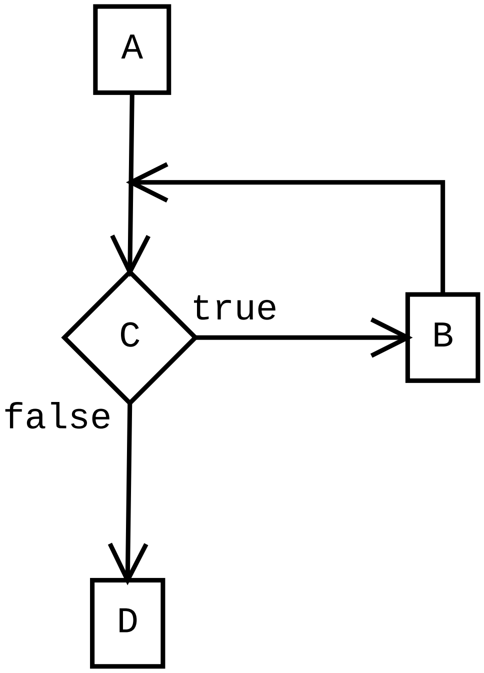
      i. the list of statements is executed,

      ii. and when finished, the whole process is repeated from step 1

Loops

```
A;
while (C) {
  B;
}
D
```

- Control flow diagram:



Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 100) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
-	-

(This table shows the values of the variables just before the statement in red is executed)

Printed:

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
1	-

Printed:

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
1	1

Printed:

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
1	1

Printed:

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
1	1

Printed:

```
1
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
1	3

Printed:

```
1
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
2	3

Printed:

```
1
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
2	3

Printed:

```
1
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
2	3

Printed:

```
1
3
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
2	5

Printed:

```
1
3
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
3	5

Printed:

```
1
3
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
3	5

Printed:

```
1
3
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
3	5

Printed:

```
1
3
5
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
3	7

Printed:

```
1
3
5
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
4	7

Printed:

```
1
3
5
```

Loops

```
int counter, number;
counter = 1;
number = 1;
while (counter <= 3) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

counter	number
4	7

Printed:

```
1
3
5
Done
```

Loops

```
int counter = 1;
int number = 1;
while (counter <= 10000) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
System.out.println(‘‘Done’’);
```

Loops

```
int maximum = Keyboard.readInt();
int counter = 1;
int number = 1;
while (counter <= maximum) {
    System.out.println(number);
    number = number + 2;
    counter++;
}
```

Loops

- A loop may not terminate

```
int maximum = Keyboard.readInt();
int counter = 1;
int number = 1;
while (counter <= maximum) {
    System.out.println(number);
    number = number + 2;
}
```

- A loop will not terminate if its condition is always true

The end