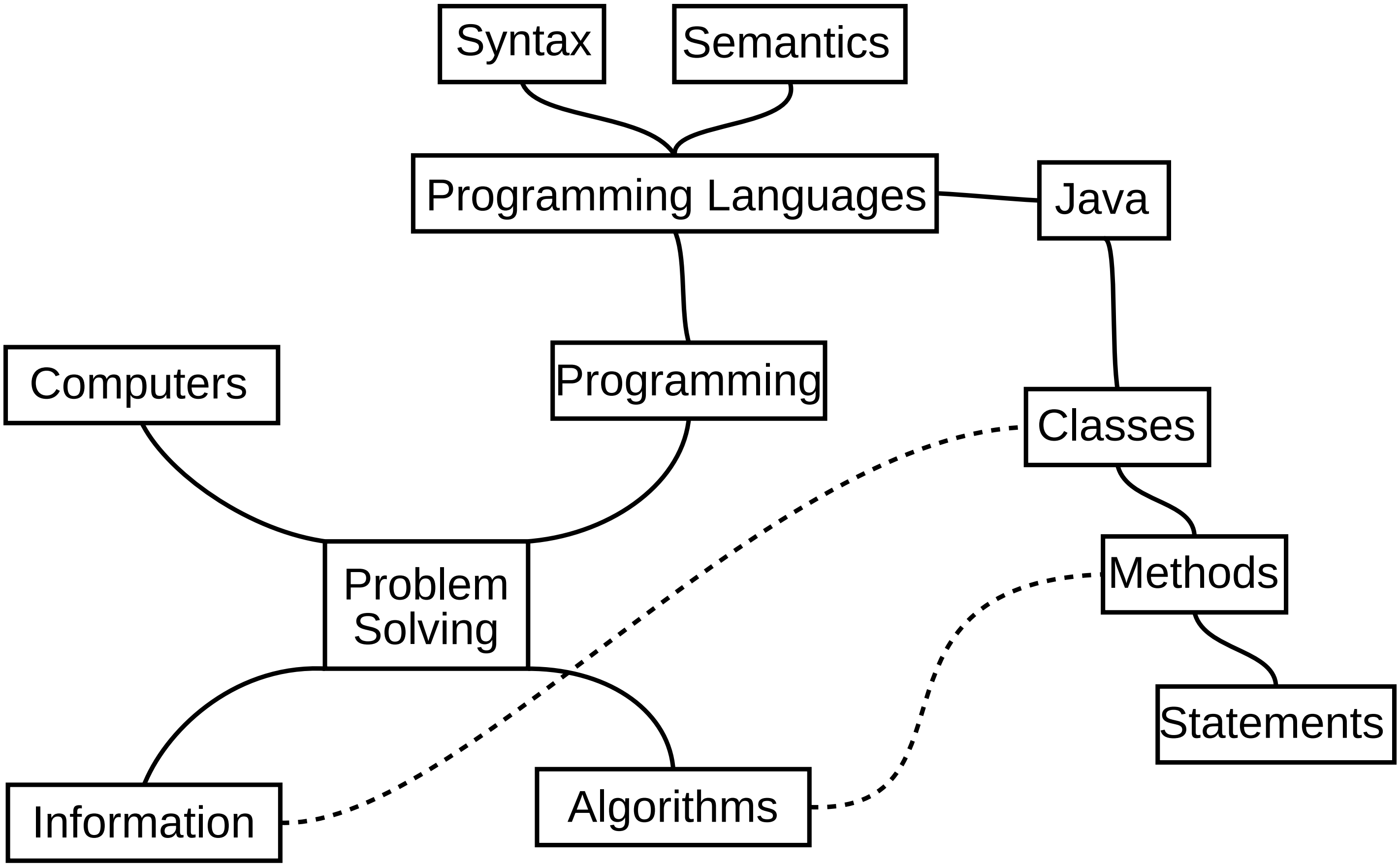
Reminder

- Wednesday's lecture will be at ENGMD 280

Road map


Syntax
Semantics
Programming Languages
Java
Computers
Programming
Classes
Problem Solving
Methods
Statements
Information
Algorithms

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

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;
}
```

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;
}
```

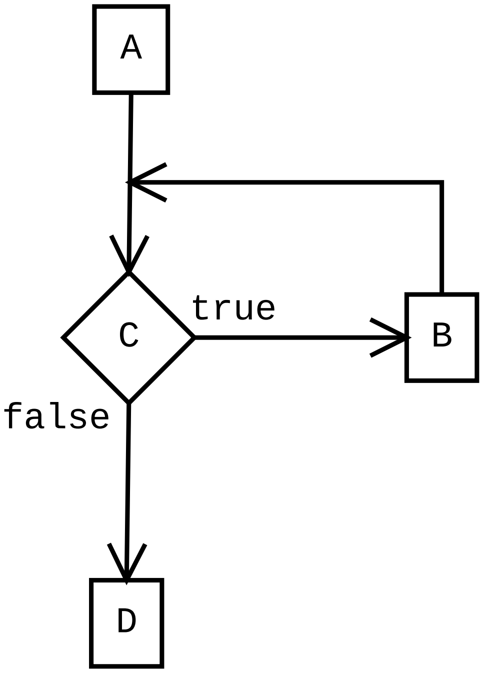
- Semantics: the execution of a while loop proceeds as follows:

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

- `while` is *not* the same as `if`

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

- The while statement executes a statement or list of statements *repeteadely*, until its condition becomes false

- The if statement executes a statement or list of statements *once*, and only if its condition is true

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 condition of a loop will remain true if its variables never change

Loops

- The variables of the condition must change in a way which eventually makes the condition false
- If the variables change, but in a way that does not make the condition false eventually, then the loop does not terminate

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

Loops

- Will this terminate?

```
int i;
i = 1;
while (i != 10) {
  //...
  i = i + 2;
}
```

Loops

- Will this terminate?

```
int i;
i = 100;
while (i != 0) {
  //...
  i = i / 2;
}
```

Loops

- Will this terminate?

```
int i;
i = 10;
while (i != 3) {
  //...
  i = i / 2;
}
```

Loops

- Will this terminate?

```
float i;
i = 10;
while (i != 0) {
  //...
  i = i / 2;
}
```

Loops

- Termination is important

Gussing game

```
import cs1.Keyboard;
public class GuessingGame {
  public static void main(String[] args)
  {
    int die, guess, points, game;
    final int ROUNDS = 10;

    points = 0;
    game = 1;
    while (game <= ROUNDS) {
      System.out.print(“What is your guess? ”);
      guess = Keyboard.readInt();

      die = (int)(Math.random() * 6 + 1);
      if (guess == die) {
        points++;
      }
      game++;
    }
    System.out.println(“You guessed ”+points+“ tim
  }
}
```

Reverse

- Problem: Given any string, print the string in reverse.

- Analysis:
  - Information involved: a four letter word, *w*.
  - Input: *w*
  - Output: a word *v* which is the reverse of *w*
  - Definitions:
    - The *reverse* of a word *w* is a word *v* which has the the same characters as *w*, but in inverse order: the first letter of *v* is the last of *w*, the second letter of *v* is the second-to-last of *w*, etc.
  - Note: no restrictions on the string!

Design

The design for only strings of size 4:

1. Obtain the word w

2. Create a new word v, initially empty

3. Add the last character of w to the end of v

4. Add the third character of w to the end of v

5. Add the second character of w to the end of v

6. Add the first character of w to the end of v

7. Print v

Design

Generalise the design:

1. Create a new word *v*, initially empty
2. Add the last character of *w* to the end of *v*
3. Add the second to last character of *w* to the end of *v*
4. ...
5. Add the second character of *w* to the end of *v*
6. Add the first character of *w* to the end of v
7. Print *v*

Design

Generalise the design:

1. Create a new word *v*, initially empty

2. Traverse the string *w* from last character to first, adding the corresponding character at the end of *v*

3. Print *v*

Design

Generalise the design:

1. Create a new word *v*, initially empty

2. Set a variable *index* to be the last index of *w*

3. While the *index* is larger or equal to 0, repeat:

    (a) Let *c* be the character at index, of the string w.
    (b) Append *c* to *v*
    (c) decrement *index* by 1

4. Print *v*

Implementation

```
// This solution traverses w from right to left
String w, v;
int index;
char c;

v = ‘‘’’;
index = w.length() - 1;
while (index >= 0) {
  c = w.charAt(index);
  v = v + c;
  index--;
}
```

Implementation

```
// This solution traverses w from left to right
String w, v;
int index;
char c;

v = ‘‘’’;
index = 0;
while (index <= w.length() - 1) {
  c = w.charAt(index);
  v = ‘‘’’ + c + v;
  index++;
}
```

Prime numbers

- Problem: determine whether a given positive integer is prime or not

- Analysis:
  - Input: an integer n
  - Output: a boolean: true if n is prime, false otherwise
  - Definitions:
    - A *prime* number is a number which is divisible only by 1 and itself
    - An integer *a* is *divisible* by *b* if there is an integer k such that $a = kb$
  - Assumptions: *n* is positive

Prime numbers

- Basic idea: try to find a factor of n (i.e. a number that divides n), between 1 and n. If such number exists. then n is not prime, otherwise it is prime.

```
1. Set is_prime to true
2. Set $i$ to be 2
3. While $i < n$, repeat:
   (a) if $i$ divides $n$, then set is_prime to false
   (b) increment i by 1
4. Return the value of is_prime
```

Prime numbers

```
boolean is_prime = true;
int i = 2;
while (i < n) {
  if (n % i == 0) is_prime = false;
  i++;
}
```

Prime numbers

```
boolean is_prime = true;
int i = 2;
while (i < n) {
  if (n % i == 0) {
    is_prime = false;
    i = n;
  }
  i++;
}
```

Prime numbers

```
boolean is_prime = true;
int i = 2;
while (i < n) {
  if (n % i == 0) {
    is_prime = false;
    break;
  }
  i++;
}
```

The end