# Coding style guidelines

## Naming

Identifiers use only ASCII letters and digits.

**Classes** – mixed case, the first letter of each word in the name will be uppercase and all other letters will be in lowercase. Class names are typically nouns or noun phrases – Character.

**Variables** – mixed case starting with lower case – line, audioSystem. is-prefix should be used for Boolean variables – isFinished.

**Methods** – contains a verb and written in mixed case starting with lower case - getName(). is-prefix should be used for Boolean methods – isSet.

Other naming conventions:

**Packages** – all lower case – mypackage.

**Types** – nouns are written in mixed case starting with upper case – Line, AudioSystem

**Constants** - all uppercase using underscore to separate words – MAX_ITERATIONS.

**Collection of objects** - Plural form should be used - Collection<Point> points, int[] values.

**Iterator variable** – should be called I, j, k etc.

**Abbreviations** – should be avoided - computeAverage(); and not compAvg();

## Formatting

**Maximum Line length** – 120 characters. Split the line when a statement exceeds this limit.

**Indentation** – All indents are four spaces. All indenting is done with spaces, not tabs. All if, while and for statements must use braces even if they control just one statement.

```
if (you.hasAnswer()) {
   you.postAnswer();
} else {
   you.doSomething();
}
```

**Spaces** – All identifiers are surrounded with whitespace. There are a few exceptions to this rule:

1. All method names should be immediately followed by a left parenthesis - foo(i, j) and not foo (i, j).
2. All array dereferences should be immediately followed by a left square bracket – args[0] and not args [0].
3. The unary operator should be immediately preceded or followed by the operand - count++.
4. The cast should be written with no spaces - (MyClass)v.get(3).

**Empty lines** - Use the occasional blank line within methods to break up related chunks of code.
Use one or two blank lines between all methods.

## Comments and documentation

**Comments** –

Tricky code should not be commented but rewritten – the code should be self-documented by appropriate name choices and an explicit logical structure.

There should be a space after the comment identifier - // This is a comment and not //This is a comment.

Comments should be indented relative to their position in the code –

```
while (true) {
  // Do something
  something();
}
```

Comments may be in /* ... */ style or // ... style. For multi-line /* ... */ comments, subsequent lines must start with * aligned with the * on the previous line.

```
/*
 * This is       // And so
 * okay.         // is this.
 */
```

Comments are not enclosed in boxes drawn with asterisks or other characters.

**Documentation** –

Write self-documenting code:

"Any fool can write code that a computer can understand.

Good programmers write code that humans can understand."

Write specification to any method in the format of:

@requirs - a paragraph describing the constraints under which the abstraction is defined.

@modifies - a paragraph that specifies the names of objects that the method may modify.

@effects - a paragraph describing the behavior of the method for all inputs not disallowed by the @requirs paragraph. Defines the outputs and changes for the objects in @modifies and cannot be ignored.

Standard Javadoc paragraphs that are sufficient for specification can also be used.

ADT specification should be written in the head and include general specification – abstract state and specification fields:

```
/**
 * ComplexNumber is an immutable class that represents any complex number in
 * form of a + bi; a and b must be real numbers. There are methods for basic manipulations.
 */
```

## Coding

**Compound** - do not compound increment or decrement operators - use an extra line for the increment or decrement.

foo(x); And not foo(x++);

x++;

**Initialization** - initialize all variables when they are declared.

**Scope** - all class attributes must always be private, except for inner classes and some final values.

4.

א.

מפרט למתודה findfirst:

@requirs arr != null

@modifies nothing

@effects returns the index of the first occurrence of val in arr. If val is not found, returns arr length.

מפרט למתודה findLast:

@requirs arr != null

@modifies nothing

@effects returns the index of the last occurrence of val in arr. If val is not found, returns -1.

ב. מפרט חזק יותר למתודה findLast:

@requirs nothing

@modifies nothing

@effects returns the index of the last occurrence of val in arr. If val is not found, returns -1. If arr == null, returns -2.

```
public static int findLast(int[] arr, int val) {
   if(arr == null)
      return -2;
   for(int i = arr.length - 1; i >= 0 ; i--)
      if(arr[i] == val)
         return i;
   return -1;
}
```