Java Programming Standard Spring 2020

1 Naming

1. Always use meaningful and preferably short names for functions and attributes.
2. The method that returns the attribute *attribute* should be called `getAttribute`. Similarly, the method that changes that attribute, should be called `setAttribute`. Methods that check if a certain property holds and return a Boolean, should start with *is*, for example `isEmpty` or `isFull`.
3. Class names always start with an uppercase letter.
4. Class names are singular nouns, for example `Person` or `Vehicle`.
5. Names of attributes and methods start with a lowercase letter. In all names, like classes, attributes, and methods, you can use uppercase letters to improve readability. The letter after the place where you would normally have whitespace becomes an uppercase letter, for example `numberOfPeople`, `isEmpty` or `CheeseSandwich`. This is called *CamelCase*.
6. Constant attribute names are written with only capital letters. If there would have been whitespace, use an underscore, for example `MAX_SIZE` or `PI_SQUARED`.
7. Use only one natural language while programming, English in this course.
8. Use `Refactor > Rename` in NetBeans to improve names of existing code without introducing errors.

2 Layout

1. All statements in a block have the same amount of indenting.
2. Statements in a compound statement are indented by one level, which consists of four spaces.
3. There are two styles of using brackets `{ }` for compound statements that you can use in this course:
   (a) The brackets around a compound statement are on their own line. The closing bracket is below the opening bracket, at the same level of indenting.
   (b) The opening bracket is on the same line as the keyword/condition. Use a space between the bracket and the keyword/condition. The closing bracket comes below the first letter of the keyword.
4. Use spaces between operators (e.g. `+, -`) and their arguments.
5. Put a newline between method definitions in classes.
6. Do not make lines longer than 80 characters.
7. Use `Source > Format` from NetBeans to achieve consistent layout in an easy way.

3 Documentation

1. Write a few lines of comments before every class definition if its role is not clear from the class name.
2. All comments should be in javadoc format. That means the comments start with `/**`.
3. Only provide comments when necessary.

4 Using the programming language Java

1. Within a class, first define the attributes, then the constructors and finally the methods.
2. Never make attributes `public`. Only `final` values may be `public`.
3. Always explicitly use an access modifier for you attributes and methods: `public`, `private`, or `protected`.
4. In every class, define at least one constructor, even though the body might be empty.
5. Use the superclass constructor `super` explicitly.
6. Initialize all attributes in the constructors.
7. Use `final static` attributes instead of magic numbers in your program.

```
// bad:
Customer[] cs = new Customer[100];
```

```
// good:
final static MAX_CUSTOMERS = 100;
Customer[] cs = new Customer[MAX_CUSTOMERS];
```

8. Methods that are only intended for internal use within a class should be `private`.
9. Throw an exception at each error situation, for example illegal arguments for a method.
10. Use iterators for collections. Use a *for each* loop if you only want to use the elements. Use an explicit iterator if you also want to change the elements.
11. Divide your program in classes in a logical way. Make the interface between classes (the methods with modifier `public`) as small as possible/reasonable.
12. Use constructs such as subclasses and interfaces to structure your program. Programs need to be clear, and not demonstrate that you are a Java wizard.
13. Have one class dedicated for the main function.

5 Example of method definition

Method definition in bad style, **DO NOT COPY THIS**.

```
int ugplypower(int a ,int b )
{int x=a,n=b ,r=1;
   while (n!=0 )
  if(n%2==0) {x=x*x;n= n/2 ; }
  else { r=r*x ;n= n-1; }
    return r;}
```

How you could write this method according to the rules above.

```
/**
 * computes {@code base} to the power {@code exp}
 *
 * @param  base value of base
 * @param  exp  value of exponent
 * @return {@code base^exp} if {@code exp >= 0}
 * @throws IllegalArgumentException if {@code exp < 0}
 */
public static int power(final int base, final int exp) {
    if (exp < 0) {
        throw new IllegalArgumentException(
            "power: exponent is negative (" + exp + ")");
    }
    // exp >= 0

    int result = 1;
    int b = base;

    // loop invariant: 0 <= n <= exp && result * b^n == base^exp
    for (int n = exp; n > 0; ) {  // n >= 0
        if (n % 2 == 0) {         // even exponent
            b = b * b;
            n = n / 2;
        } else {                  // odd exponent
            result = result * b;
            n = n - 1;
        }
    }
    // n == 0, hence result == base^exp using the invariant
```

```
        return result;
    }
```