# Software Development (CS2514) Assignment 3

Interfaces and Generics (Due: April 14th. Marks: 10)

## General Comments

Carefully read the submission guidelines before you submit the assignment.

Like all other exercises, this is an exercise about implementing *maintainable* classes and interfaces. You should always assume the specifications may change (slightly) and make sure your implementation can implement these changes with the minimum amount of effort.

Before you start implementing your classes, please make sure you understand the API. If you don't you'll make your life much more difficult.

**This assignment overrides a previously posted assignment which had the same number but this assignment is different and carries more marks. Some elements of the assignment are similar to the previously posted assignment but this assignment is technically more challenging. I suggest you start implementing this assignment from scratch, even if you already have a partial/complete implementation of the previous version of assignment.**

## Learning Objectives

For this assignment you will learn about interfaces and generics. You will learn how to implement a hierarchy for `Bikes` which can enumerate their `Components`. The hierarchy is slightly simpler than the hierarchy of Assignment 2. However, this time you will learn how to implement the hierarchy without inheritance. The following are the main required techniques.

- The interfaces define the hierarchy.
- Each class is `final` and cannot be extended.
- For every well-defined behaviour, you will implement a dedicated concrete class which defines the behaviour. Please note that `enum` classes are not allowed.
- You will re-use the implementation of your concrete classes in other (concrete) classes using delegation.[1]

### Main Details

In this assignment you will implement a `Bike` hierarchy which has the `Bike` interface sitting at the top of the hierarchy.

#### The `Bike` Interface

The `Bike` interface represents bikes which can enumerate their `Components` using the `Iterable` interface. There are three kinds of `Bikes`: `MountainBike`, `CityBike`, and `Hybrid`. All `Bikes` have a `Frame`, which is a `Component`.

[1]Delegation is covered in Lectures 8 and 9.

A `Bike` should be able to enumerate its `Components` using the `Iterable` interface. E.g.

```
final MountainBike bike = OMITTED;

for (Component component : bike) {
    USE component;
}
```

A `Bike` can print its `Components` with two methods:

- `void print( )`, which prints the `Components`. The `Components` are printed on the same line but separated with a dedicated separator symbol, which is given by `", "`. For example, when a `CityBike` prints itself with this method it may print the following.

  ```
  high frame, front light: off, rear light: off, carrier
  ```

  Please note that `Components` print themselves in normal English, that the `Lights` print their names and their on-off status, and there isn't a comma at the end of the line.
- `void println( )`, which prints the `Components`. The `Components` are printed but separated with a dedicated separator symbol, which is given by the new line symbol. For example, when a `CityBike` prints itself with this method it may print the following.

  ```
  high frame
  front light: off
  rear light: off
  carrier
  ```

- Hint: if you are clever, you can implement the previous two methods using a single auxiliary method, which enumerates the `Bike`'s `Components`.

### The `Component` Interface

`Component` is an interface which should define the following abstract method:

- `public void print( )`: print the `Component`;

A `Light` is a special `Component` which can be turned on and off. When a `Light` prints itself, it also prints the state of the light (on or off).

Components should print themselves as *proper* English. For example, a `MediumFrame` instance should not print `MediumFrame` but it should print `medium frame` or equivalent.

### Further `Bike` Details

When a bike has `Lights`, the bike should have a method which turns the `Lights` on and a method which turns the `Lights` off.

The following are the concrete `Bike` classes and their `Components`.

**`MountainBike`:** `LowFrame`.

**`CityBike`:** `HighFrame`, `Carrier`, `FrontLight`, and `RearLight`.

**`Hybrid`:** `MediumFrame`, `FrontLight`, and `RearLight`.

Your concrete `Bike` classes will need to override the method `iterate( )` at some stage. Your may use an anonymous class to do this. Besides this you may **not** use nested classes , anonymous classes, and nested interfaces;

### The `Main` Class

Please implement a small `Main` class, which defines a `main` method, which creates the bikes and prints them.

### Hints

The following are some hints, which are not exhaustive.

- Define a `Bike` and a `Component` interface.
- Define a `ConcreteComponent` class which overrides the abstract `Component` methods.
- Define a `ConcreteBike` class which overrides the abstract `Bike` method for bikes with a frame.
- Re-use the behaviour which is implemented in the `ConcreteBike` and `ConcreteComponent` classes in other concrete `Bike` and `Component` classes. You can do this with delegation.
- Every named class and every interface `must` be defined in its own file.
- …

The majority of your methods should consist of a single statement, which forwards its task to a delegate.

Please note the following.

- You may **not** use packages but you may use the `Java` libraries in `java.lang` and `java.util`;
- All attributes should be `private` and all classes should be `final`;
- Besides a comment which lists your name and ID, you don't have to provide any other comments.

## Submission Details

- Remember that each class/interface should be in its own file. You will lose marks if you violate this rule.
- Each class and interface should have a comment which provides your name and student ID.
- **As an exception (for this assignment only) you don't have to provide any other comments.**
- Use the CS2514 `Canvas` section to upload your classes as a single *.tgz* archive called *Lab-3.tgz* before 23.55 p.m., April 14th, 2020. To create the *.tgz* archive, do the following:
  - ⋆ Create a directory `Lab-3` in your working directory.
  - ⋆ Copy `Bike.java` and your other user-defined Java files into the directory. Do not copy any other files into the directory.
  - ⋆ Run the command '`tar cvfz Lab-3.tgz Lab-3`' from your working directory. The option 'v' makes `tar` very chatty: it should tell you exactly what is going into the `.tgz` archive. Make sure you check the `tar` output before submitting your archive.
  - ⋆ Note that file names in `Unix` are case sensitive and should not contain spaces.
- Note that the format of your submission should be `.tgz`: do *not* submit `zip` files, do *not* submit `tar` files, do *not* submit `bzip` files, and do *not* submit `rar` files. If you do, it may not be possible to unzip your assignment.
- No marks shall be awarded for classes that do not compile.