Design patterns are a set of reusable, proven solutions to common design problems. Micro patterns are specific, low-level solutions.

```java
public class Location {
    public final int line;
    public final int column;
    public final int offset;
    public final String fileName;

    public Location(String fileName, int offset, int line, int column) {...
}

    @Override public String toString() {...
    @Override public boolean equals(Object other) {...
    @Override public int hashCode() {...

    features:
    ...}
}
```

Observer pattern is a behavioral design pattern that enables an object to maintain a list of its dependents (or observers) and to notify them automatically of any events that happen to it.

```c++
struct person : string name, int year;
struct woman, superwoman;
struct man : person { woman *spouse; }
struct woman : person { man *spouse; }
struct super : int speed;
struct superman : man, super { superwoman *partner; }
struct superwoman : woman, super { superman *partner; }
```

In C++, the `super` struct is a superset of `man` and `woman`, and `superman` and `superwoman` are classes that can have an instance of the `super` struct as a member variable.

In Java, the `Location` class is a simple data structure that stores line, column, offset, and file name information.

In micro patterns, a single pattern is often used to solve a specific problem, while design patterns are broader and more general solutions to common design problems.

In an enterprise application, patterns can be used to improve code quality, maintainability, and scalability.

In micro patterns, a single pattern is often used to solve a specific problem, while design patterns are broader and more general solutions to common design problems.
abstract class Algorithm { public abstract void run(Data d); }

class CountEmpty extends Algorithm {
  @Override public void run(Data d) {
    int result = 0;
    for (String s : d.items)
      if (s == null || s.length() == 0)
        result++;
    System.out.println(result);
  }
}

class MaxLen extends Algorithm {
  @Override public void run(Data d) {
    int result = 0;
    for (String s : d.items)
      result = Math.max(result, s.length());
    System.out.println(result);
  }
}

class Data {
  public final Vector<String> items = new Vector<String>();
  public void add(String s) { items.add(s); }
  public static void main(String[] args) {
    Data d = new Data();
    d.add("abc");
    d.add(null);
    d.add("def");

    new CountEmpty().run(d);
    new MaxLen().run(d);
  }
}

public class MyProgram {
  private final String[] args_;
  public MyProgram(String[] args) { args_ = args; }
  private void readFile() {
  ... }
  private void buildData() {
  ... }
  private void sortData() {
  ... }
  private void printRecords() {
  ... }

  public void run() {
    readFile();
    buildData();
    sortData();
    printRecords();
  }

  public static void main(String[] args) {
    new MyProgram(args).run();
  }
}