1.

a. Class BloomFilterDet:

```
      public BloomFilterDet(int setSize, int bitsPerElement):
constructor to create bloomfilter andcalculate number of hash functions
required along with their respective Random Values (explained later)

        public long[] hashvalue(String S1):
returns an array of  hash values for string S1.

        public void add(String S2):
converts String to lower case and sets bit corresponding to all hash values
as true
        public boolean appear(String S):
converts String to lower case and checks if all hashvalues are 1. If yes it
interprets it as the String is present in bloom filter

        public int filterSize():
It returns total size of bloomfilter created

        public int dataSize():

returns number of element added.
```

b. Class BloomFilterRan:

```
      public BloomFilterDet(int setSize, int bitsPerElement):
constructor to create bloomfilter and calculate number of hash functions,
prime p required and their respective a and b values (explained later)

        public long[] hashvalue(String S1):
returns an array of  hash values for string S1.

        public void add(String S2):
converts String to lower case and sets bit corresponding to all hash values
as true
        public boolean appear(String S):
converts String to lower case and checks if all hashvalues are 1. If yes it
interprets it as the String is present in bloom filter

        public int filterSize():
It returns total size of bloomfilter created

        public int dataSize():

returns number of element added.
```

c. Class FalsePositives:

```
        public static void main(String args[]):
It runs test for arbitrary 200 element array over both Deterministic and
Random hash vales.

        public static void CheckDet():
Runs test over Deterministic hash value and display false positive rates
        public static void CheckRan():
Runs test over random hash value and display false positive rates
```

d. Class BloomDifferential:
public static void main(String args[])
Start of program. It calls method createFilter to create bloom filter and then takes input of key from user. It passes this to method retrieveRecord.

public static void createFilter():
Retrieves key values from DiffFile and adds it to bloom filter.

public static String retrieveRecord(String Key1):
Searches for hash value of key. If found searched in DiffFile and returns record. If not found reports as false positive and searches in database. If hash value not found then searches directly in database.

e. Class NaiveDifferential:
public static void main(String args[])
Start of program. It takes input of key from user. It passes this to method retrieveRecord.

public static String retrieveRecord(String Key1):
Searches for key in DiffFile. If not found searches in database.

f. Class Empirical Comparison:
Returns runtime of BloomDifferential and NaiveDifferential after testing over all keys in gram.txt.

2.
For Class BloomFilterDet:
To generate k hash values the program creates k random values. It multiplies this random value to initial value of hash. Ie
Hash value=FNV * random value.
This allows us to start from random values and hence gives k different hash values to a given string

3.
For Class BloomFilterRan:
To generate k hash values we randomly choose k (a,b) pairs and store them. Both a and b are less than prime p. the hash value is then calculated as (ax+b)%p.

4.
Experiment to compute False positive:
In this experiment we first create a list of 200 random elements. These elements have values ranging from 0-100.
Then we create a similar random list of 100 elements.
Now, remove every occurrence of elements in second list from the first list. Hence we are left with a sure list of elements that are not present in first list.
We create a hash for first list and check hash for every element in second list. If it is present then it is considered a false positive.
False positive rate is then calculated as number of false positives / false positives+data elements in the bloom filter (returned by dataSize())

5.

BloomFilterDet:

Bits Per Element 4: 0.359
Bits Per Element 8: 0.185
Bits Per Element 10: 0.112

BloomFilterRan:
Bits Per Element 4: 0.431
Bits Per Element 8: 0.227
Bits Per Element 10:0.14

Theoretically false positive values= 0.618^bits per elemt.
Thus,
Bits Per Element 4: 0.1
Bits Per Element 8: 0.02
Bits Per Element 10:0.008

The experiment displays that the experimental probabilities are greater than the theoretical values. We also see that the false probability rate decreasing as the bits per element increases. It also shows that FloomFilterDet has smaller value as compared to BloomFilterRan hence it is a much better choice of hash function.

6.

Empirical Comparison:

First a bloom filter is created
Keys are collected from gram.txt and searched in the bloom filter.

We record Start time and End time of both Search methods.
Notice we create filter twice since we subtract time taken to create such a filter.

PLEASE NOTE:
Theoretically using a bloom filter is much more efficient. However, in the experiment there are false positives being found which does not represent actual efficiency.

While coding there was care taken to follow the theoretical methodology as much as possible but the implementation of the program has resulted in false positives for every new key.