5.1 An interface is such that it contains state names and methods, but the methods must be undefined. Therefore cannot exist as an object, but are templates for other classes. Multiple interfaces can be implemented by classes.

An Abstract class contains variables, methods and abstract methods. Abstract classes can only exist as objects if another class extends them. Classes can only extend one abstract class at a time.

A normal class can contain variables and methods. They cannot contain abstract methods. Classes can exist as objects and they also can be extended by other classes.

5.7

a)

interface Queue {

public int takeHead() throws Exception;

public void addTo(int x);

}

b)

public class OOPListQueue implements Queue {

private OOPLinkedList List1;

private OOPLinkedList List2;

public OOPListQueue() {

List1 = new OOPLinkedList();

List2 = new OOPLinkedList();

}

@Override

public int takeHead() throws Exception {

if (List1.length() == 0) {

Normalise();

}

return List1.remove();

}

@Override

public void addTo(int x) {

List2.add(x);

}

public void Normalise() throws Exception {

if (List2.length == 0) {

throw new Exception("Queue is Empty");

} else {

for(int i = 0;i < List2.length();i++) {

List1.add(List2.remove());

}

}

}

}

c)

public class OOPArrayQueue implements Queue {

private OOPArrayList mQueue;

private int head;

private int end;

public OOPArrayQueue() {

head = 0;

end = 0;

mQueue = new OOPArrayList();

}

@Override

public int takeHead() throws Exception {

if (end-head == 0) {

throw new Exception("Empty Queue");

} else {

return mQueue.get(head++);

}

}

@Override

public void addTo(int x) {

mQueue.add(end++,x);

}

}

d) for b O(n), for c O(1)

Exceptions:

Checked Exceptions are those which are declared in a method, and so the object calling the method must deal with the exceptions thrown from the method, they are expected to be handled by program. Eg public void X() throws SomeException {}.

Unchecked Exceptions are those which are not expected to be handled. These will typically be Programming Errors. Eg NullPointerException.

9.4

package uk.ac.cam.jsc89.oop.SupervisionWork;

public class MyClass implements Cloneable {

private String mName;

private int[] mData;

// Copy constructor

public MyClass(MyClass toCopy) {

this.mName = toCopy.mName;

this.mData = new int[mData.length];

for (int i = 0;i<toCopy.mData.length;i++){

this.mData[i]=toCopy.mData[i];

}

}

//Clone

@Override

public Object clone() throws CloneNotSupportedException {

MyClass clone = (MyClass)super.clone();

for (int i = 0;i<this.mData.length;i++){

clone.mData[i]=this.mData[i];

}

return clone;

}

}

c) If you want to copy an object which has subclasses, you are unable to copy this unless all objects contain the subclasses. Therefore the copy constructor loses information in the copy.

d) If we don’t know the make up of the rest of the code, it could become difficult to cast the clone. Copy Constructors do not require knowledge of the whole hierarchy therefore they are prefable.