3 Object-Oriented Programming (rkh23)

The Java Collections framework contains `PriorityQueue<E>`, which represents a priority queue based on a priority heap of objects of type `E`. Priority is specified implicitly by providing an ordering on `E` via either the `Comparable` or `Comparator` interfaces. Consider a task tracking app that uses the following class to represent a task:

```
public class WorkTask {
   private int priority;
   private String description;

   public WorkTask(String descriptor, int priority) {
      this.descriptor = descriptor;
      this.priority = priority;
   }

   public int getPriority() { return priority; }
   public int setPriority(int priority) {this.priority = priority; }
}
```

(*a*) (*i*) Compare and contrast Comparable and Comparator in Java's Collections framework. [3 marks]

(*ii*) Show how to make a `PriorityQueue<WorkTask>` maintain its contents highest priority first using:

(A) `Comparable` [2 marks]

(B) `Comparator` [2 marks]

(*iii*) Would you prefer Comparable or Comparator for this application? Explain your answer. [1 mark]

(*b*) PriorityQueue does not offer a method to change priorities. Instead, an object with a changed priority must be removed and reinserted. Using a design pattern that you should specify, write code for `AutoUpdatableQueue`, which extends `PriorityQueue` and automatically updates the queue when the priority of any object in the queue is updated. Make your solution flexible and demonstrate how to apply it to a queue of `WorkTask`s. [12 marks]