Solutions

Solution 1.

```
import java.util.LinkedList;

public class SalleDAttentePAPS extends LinkedList implements SalleDAttente {

  private int var_capacite;

  public SalleDAttentePAPS(int c) {
    super();
    if (c <= 0)
      throw new RuntimeException("Taille_non_valide");
    var_capacite = c ;
  }

  // Nombre de clients dans la salle
  public int capacite() {
    return var_capacite;
  }

  // Nombre de clients dans la salle
  public int nbClientsEnAttente()       {
    return size() ;
  }

  // Salle vide ?
  public boolean estVide() {
    return isEmpty() ;
  }

  // Salle pleine ?
  public boolean estPleine() {
    return nbClientsEnAttente() == capacite() ;
  }

  // Prochain clientà servir
  public Object prochain() {
    if (estVide())
      throw new RuntimeException("Salle_vide");
    return getFirst();
  }

  // Sortie de la salle du prochain clientà servir
  public void sortir() {
```

```
    if (estVide())
      throw new RuntimeException("Salle vide");
    removeFirst() ;
  }

  // éEntre dans la salle du client c
  public void entrer(Object o) {
    if (estPleine())
      throw new RuntimeException("Salle pleine");
    addLast (o) ;
  }
}
```

Solution 2.

```
import java.util.LinkedList;

public class SalleDAttenteAvecPrio implements SalleDAttente {

  private int var_prioriteMaximum ;
  private int var_nbClientsAttente ;
  private int var_capacite ;
  private LinkedList[] tabSalles ;

  public SalleDAttenteAvecPrio(int c, int prioMax) {
    if (c <= 0 || prioMax < 0)
      throw new RuntimeException("Arguments non valides");
    var_capacite = c ;
    tabSalles = new LinkedList[prioMax+1] ;
    var_prioriteMaximum = prioMax ;
    for (int i = 0; i <= prioMax ; i++) {
      tabSalles[i] = new LinkedList() ;
    }
  }

  public int maxPrio() {
    return var_prioriteMaximum ;
  }

  public int capacite() {
    return var_capacite ;
  }

  public int nbClientsEnAttente() {
    return var_nbClientsAttente ;
  }
```

```
public boolean estVide() {
  return nbClientsEnAttente() == 0 ;
}

public boolean estPleine() {
  return nbClientsEnAttente() == capacite() ;
}

public Object prochain() {
  if (estVide())
    throw new RuntimeException("Salle vide");
  return salleNonVidePrioritaire().getFirst ();
}

public int prioriteProchain() {
  if (estVide())
    throw new RuntimeException("Salle vide");
  return ((AvecPrio) prochain()).getPrio() ;
}

public void sortir() {
  if (estVide())
    throw new RuntimeException("Salle vide");
  salleNonVidePrioritaire ().removeFirst();
  var_nbClientsAttente -= 1 ;
}

public void entrer(Object c) {
  if (estPleine ())
    throw new RuntimeException("Salle pleine");
  AvecPrio objetAvecPrio = (AvecPrio) c ;
  sousSalleAvecPrio(objetAvecPrio.getPrio()).addLast(c) ;
  var_nbClientsAttente += 1 ;
}

private LinkedList sousSalleAvecPrio (int prio) {
  if (prio < 0)
    throw new RuntimeException("Priorite non valide");
  if (prio > maxPrio()) {
    prio = maxPrio() ;
  }
  return tabSalles[prio] ;
}

private LinkedList salleNonVidePrioritaire() {
```

```
    if (nbClientsEnAttente() == 0) {
      return null;
    }
    int i = maxPrio() ;
    while (i > 0 && sousSalleAvecPrio(i).isEmpty()) {
      i-- ;
    }
    return sousSalleAvecPrio(i) ;
  }
}
```