```
 System.out.println();
 System.out.println();
 System.out.println();}
```

puis l'utiliser dans le programme principal

```
System.out.print("Le RER en direction de ");
System.out.print("Saint-Rémy-lès-Chevreuse");
System.out.print(" partira à ");
System.out.println("8h50");
sauterTroisLignes();

System.out.print("Le RER en direction de ");
System.out.print("Massy-Palaiseau");
System.out.print(" partira à ");
System.out.println("8h55");
sauterTroisLignes();
```

L'instruction `sauterTroisLignes();` que l'on trouve dans le programme principal se nomme l'*appel* de la fonction `sauterTroisLignes`. L'instruction que l'on isole du programme dans une fonction et que l'on exécute à chaque appel s'appelle le *corps* de la fonction.

Organiser un programme en fonctions permet d'éviter les redondances. En outre, cela rend les programmes plus clairs et plus faciles à lire : pour comprendre le programme ci-dessus, il n'est pas nécessaire de savoir comment la fonction `sauterTroisLignes();` est programmée, il suffit de savoir ce qu'elle fait. Enfin, cela permet d'organiser l'écriture du programme. On peut décider d'écrire la fonction `sauterTroisLignes();` un jour et le programme principal le lendemain. On peut aussi organiser une équipe de manière à ce qu'un programmeur écrive la fonction `sauterTroisLignes();` et un autre le programme principal.

Ce mécanisme peut se comparer au mécanisme des définitions du langage mathématique qui permet d'utiliser le mot « groupe » au lieu de répéter la locution « ensemble muni d'une loi interne, associative, ayant un élément neutre, dans lequel tout élément a un symétrique ».

2.1.2 Le passage d'arguments

Certains langages de programmation, comme les assembleurs ou Basic, n'ont qu'un procédé sommaire d'abréviations, comme celui-ci. Mais l'exemple ci-dessus montre que ce mécanisme est insuffisant pour éviter les redondances, car le programme initial est composé de deux copies, quasiment identiques, du