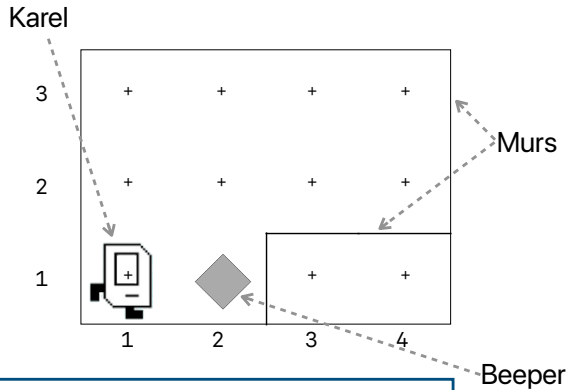
Karel est un simple robot vivant dans un monde simple. En donnant des commandes a Karel, vous pouvez le diriger pour accomplir certaines tâches dans son monde. Le processus de spécification de ces commandes est appelé la programmation. Initialement, Karel ne comprends qu'un nombre limité de commandes prédéfinies. Mais vous pouvez apprendre de nouvelles commandes a Karel afin d'étendre ses capacités.



Commandes et conditions de Karel

	Commande
Avancer	`move();`
Tourner à gauche	`turnLeft();`
Ramasser beeper	`pickBeeper();`
Déposer beeper	`putBeeper();`

Test	Test Opposé	Description
`beepersPresent()`	`noBeepersPresent()`	Au moins un beeper sur la cellule?
`beepersInBag()`	`noBeepersInBag()`	Au moins un beeper dans le sac?
`frontIsClear()`	`frontIsBlocked()`	Pas de mur devant?
`leftIsClear()`	`leftIsBlocked()`	Pas de mur à gauche?
`rightIsClear()`	`rightIsBlocked()`	Pas de mur à droite?
`facingEast()`	`notFacingEast()`	Est-ce que Karel fait face à l'Est?
`facingWest()`	`notFacingWest()`	Est-ce que Karel fait face à l'Ouest?
`facingNorth()`	`notFacingNorth()`	Est-ce que Karel fait face au Nord?
`facingSouth()`	`notFacingSouth()`	Est-ce que Karel fait face au Sud?

Additions de SuperKarel

	Commande
Tourner à droite	`turnRight();`
Faire demi-tour	`turnAround();`
Colorer cellule	`paintCorner(couleur);`

Test	Description
`random()`	Est vrai 50% du temps, mais de manière aléatoire
`random(p)`	Est vrai `p`*100% du temps, mais de manière aléatoire
`cornerColorIs(c)`	Est ce que la couleur de la cellule est `c`

Structure d'un programme Karel

```
// Importation des classes
import stanford.karel.Karel;

/*definition d'une nouvelle classe*/
public class NomDeLaClasse extends Karel {

   public void run() {
      // Instructions de la méthode ici
   }

   // Definitions de méthodes additionnelles ici
}
```

Definition de méthodes

```
private void nomDeMethode() {
  // instructions de la méthode
}
```

Structures Conditionelles

```
if (condition) {
  // instructions si condition est vraie
}
```

```
if (condition) {
  // instructions si condition est vraie
}
else
{
  // instructions si condition est fausse
}
```

Structures Itératives

```
while(condition) {
  // instructions à répéter
  // tant que condition est vraie
}
```

```
for(int i=0; i<N; i++) {
  // instructions à répéter N fois
}
```