nextForwardPosition

```
static Coordinates nextForwardPosition(Coordinates position, Direction direction) {
    if (direction == NORTH)
        return new Coordinates(position.getX(), position.getY() - 1);
    if (direction == SOUTH)
        return new Coordinates(position.getX(), position.getY() + 1);
    if (direction == EAST)
        return new Coordinates(position.getX() + 1, position.getY());
    return new Coordinates(position.getX() - 1, position.getY());
}
```

letsGo

```
 public List<CheckPoint> letsGo() throws UnlandedRobotException, UndefinedRoadbookException,
InsufficientChargeException, LandSensorDefaillance, InaccessibleCoordinate {
    if (roadBook == null) throw new UndefinedRoadbookException();
    List<CheckPoint> mouchard = new ArrayList<CheckPoint>();
    while (roadBook.hasInstruction()) {
        Instruction nextInstruction = roadBook.next();
        if (nextInstruction == FORWARD) moveForward();
        else if (nextInstruction == BACKWARD) moveBackward();
        else if (nextInstruction == TURNLEFT) turnLeft();
        else if (nextInstruction == TURNRIGHT) turnRight();
        CheckPoint checkPoint = new CheckPoint(position, direction, false);
        mouchard.add(checkPoint);
        blackBox.addCheckPoint(checkPoint);
    }
    return mouchard;
}
```

calculateRoadBook - version 1

```
static RoadBook calculateRoadBook(LandSensor sensor, Direction direction, Coordinates position, Coordinates destination, List<Instruction> instructions, List<Coordinates> trace)
        throws LandSensorDefaillance, UndefinedRoadbookException {
    if (position.equals(destination)) return new RoadBook(InstructionListTool.compacte(instructions));
    List<Direction> directionList = new ArrayList<Direction>();
    if (destination.getX() < position.getX()) directionList.add(WEST);
    if (destination.getX() > position.getX()) directionList.add(Direction.EAST);
    if (destination.getY() > position.getY()) directionList.add(Direction.SOUTH);
    if (destination.getY() < position.getY()) directionList.add(Direction.NORTH);
    List<Direction> directionsToExplored = new ArrayList<Direction>(Arrays.asList(NORTH, SOUTH, EAST,WEST));
    directionsToExplored.remove(Direction.oppositeDirection(direction));
    directionList.remove(Direction.oppositeDirection(direction));
    while (!directionsToExplored.isEmpty()) {
        if (directionList.contains(direction) && sensor.isAccessible(nextForwardPosition(position, direction)) && !trace.contains(nextForwardPosition(position,direction))) {
            try {
                directionsToExplored.remove(direction);
                directionList.remove(direction);
                Coordinates nextPos = nextForwardPosition(position, direction);
                return calculateRoadBook(sensor, direction, nextPos, destination, InstructionListTool.concatene(instructions, FORWARD), InstructionListTool.concatene(trace, nextPos));
            } catch (UndefinedRoadbookException e) {
            }
        } else if (directionList.contains(direction)) { // !sensor.isAccessible(nextForwardPosition(position, direction)
            directionsToExplored.remove(direction);
            directionList.remove(direction);
            instructions.add(TURNRIGHT);
            direction = clockwise(direction);
        } else if (!directionList.isEmpty()) {
            instructions.add(TURNRIGHT);
            direction = clockwise(direction);
        } else if (directionList.isEmpty() && directionsToExplored.contains(direction) && sensor.isAccessible(nextForwardPosition(position, direction))
                                &&!trace.contains(nextForwardPosition(position,direction))) {
            try {
                directionsToExplored.remove(direction);
                Coordinates nextPos = nextForwardPosition(position, direction);
                return calculateRoadBook(sensor, direction, nextPos, destination, InstructionListTool.concatene(instructions, FORWARD), InstructionListTool.concatene(trace, nextPos));
            } catch (UndefinedRoadbookException e) {
            }
        } else if (directionList.isEmpty() && directionsToExplored.contains(direction)) {
            directionsToExplored.remove(direction);
            instructions.add(TURNRIGHT);
            direction = clockwise(direction);
        } else if (directionList.isEmpty()) {
            instructions.add(TURNRIGHT);
            direction = clockwise(direction);
        }
    }
    throw new UndefinedRoadbookException();
}
```

calculateRoadBook - version 2

```
static RoadBook calculateRoadBook(LandSensor sensor, Direction direction, Coordinates position, Coordinates destination, List<Instruction> instructions, List<Coordinates> trace)
        throws LandSensorDefaillance, UndefinedRoadbookException {
    if (position.equals(destination)) return new RoadBook(InstructionListTool.compacte(instructions));
    List<Direction> privilegeDirections = computePrivilegeDirection(position, destination);
    List<Direction> directionsToExplored = new ArrayList<Direction>(Arrays.asList(NORTH, SOUTH, EAST, WEST));
    directionsToExplored.remove(Direction.oppositeDirection(direction));
    privilegeDirections.remove(Direction.oppositeDirection(direction));
    while (!directionsToExplored.isEmpty()) {
        if (privilegeDirections.contains(direction) || privilegeDirections.isEmpty()) {
            privilegeDirections.remove(direction);
            if (directionsToExplored.contains(direction)) {
                directionsToExplored.remove(direction);
                Coordinates nextPos = nextForwardPosition(position, direction);
                if (sensor.isAccessible(nextPos) && !trace.contains(nextPos)) {
                    try {
                        return calculateRoadBook(sensor, direction, nextPos, destination, concatene(instructions, FORWARD), concatene(trace, nextPos));
                    } catch (UndefinedRoadbookException e) {
                    }
                }
            }
        }

        instructions.add(TURNRIGHT);
        direction = clockwise(direction);

    }

    throw new UndefinedRoadbookException();
}

static List<Direction> computePrivilegeDirection(Coordinates position, Coordinates destination) {
    List<Direction> privilegeDirections = new ArrayList<Direction>();
    if (destination.getX() < position.getX()) privilegeDirections.add(WEST);
    if (destination.getX() > position.getX()) privilegeDirections.add(Direction.EAST);
    if (destination.getY() > position.getY()) privilegeDirections.add(Direction.SOUTH);
    if (destination.getY() < position.getY()) privilegeDirections.add(Direction.NORTH);
    return privilegeDirections;
}
```

lireCoordonnee

```
static Coordinates lireCoordonnee(Scanner scanner) {
    boolean conforme;
    int x = 0;
    int y = 0;
    do {
        conforme = true;
        String line = scanner.nextLine();
        String[] tokens = line.replace("(", "").replace(")", "").split(",");
        if (tokens.length != 2) {
            conforme = false;
            System.out.println("Format incorrect. c, l ou (c, l)");
        }
        else
            try {
                x = Integer.valueOf(tokens[0].trim());
                y = Integer.valueOf(tokens[1].trim());
            } catch (NumberFormatException e) {
                conforme = false;
            }
    } while (!conforme);
    return new Coordinates(x, y);
}
```

cartographier

```
public void cartographier(Coordinates landPosition) throws LandSensorDefaillance {
    if (boxTop == null)
        boxTop = new Coordinates(landPosition.getX() - PORTEE, landPosition.getY() - PORTEE);
    else if (boxTop.getX() > landPosition.getX() - PORTEE && boxTop.getY() > landPosition.getY() - PORTEE)
        boxTop = new Coordinates(landPosition.getX() - PORTEE, landPosition.getY() - PORTEE);
    else if (boxTop.getX() > landPosition.getX() - PORTEE)
        boxTop = new Coordinates(landPosition.getX() - PORTEE, boxTop.getY());
    else if (boxTop.getY() > landPosition.getY() - PORTEE)
        boxTop = new Coordinates(boxTop.getX(), landPosition.getY() - PORTEE);
    if (boxBottom == null)
        boxBottom = new Coordinates(landPosition.getX() + PORTEE, landPosition.getY() + PORTEE);
    else if (boxBottom.getX() < landPosition.getX() + PORTEE && boxBottom.getY() < landPosition.getY() + PORTEE)
        boxBottom = new Coordinates(landPosition.getX() + PORTEE, landPosition.getY() - PORTEE);
    else if (boxBottom.getX() < landPosition.getX() + PORTEE)
        boxBottom = new Coordinates(landPosition.getX() + PORTEE, boxBottom.getY());
    else if (boxBottom.getY() < landPosition.getY() + PORTEE)
        boxBottom = new Coordinates(boxBottom.getX(), landPosition.getY() + PORTEE);
    for (int i = landPosition.getX() - PORTEE; i < landPosition.getX() + PORTEE + 1; i++) {
        for (int j = landPosition.getY() - PORTEE; j < landPosition.getY() + PORTEE + 1; j++) {
            lazyGet(new Coordinates(i, j));
        }
    }
}
```

compacte

```
static List<Instruction> compacte(List<Instruction> instructions) {
    int cpt = 0;
    List<Instruction> copieCompacte = new ArrayList<Instruction>();
    for (int i = 0; i < instructions.size(); i++) {
        if (instructions.get(i) != TURNRIGHT) {
            if (cpt != 0) {
                if (cpt == 3)
                    copieCompacte.add(TURNLEFT);
                else {
                    while (cpt > 0) {
                        copieCompacte.add(TURNRIGHT);
                        cpt--;
                    }
                }
                cpt = 0;
            }
            copieCompacte.add(instructions.get(i));

        } else {
            cpt++;
        }
    }
    return copieCompacte;
}
```

carte

```
public List<String> carte() {
    List<String> carteEncadre = new ArrayList<String>();
    List<String> carte = landSensor.carte();
    Coordinates top = landSensor.getTop();
    StringBuilder positionColonne = new StringBuilder();
    positionColonne.append('\t').append('\t');
    for (int i = top.getX(); i < position.getX(); i++) {
        positionColonne.append('\t').append(i);
    }
    positionColonne.append('\t').append('\u25BC');
    for (int i = position.getX() + 1; i <= landSensor.getXBottom(); i++) {
        positionColonne.append('\t').append(i);
    }
    carteEncadre.add(carte.get(0));
    carteEncadre.add(positionColonne.toString());
    for (int i = 1; i < carte.size(); i++) {
        if (top.getY() - 1 + i == position.getY())
            carteEncadre.add("\u25B6\t" + carte.get(i));
        else
            carteEncadre.add("\t" + carte.get(i));
    }
    return carteEncadre;
}
```

main

```
public static void main(String[] args) {
    System.out.println("Consommation de base du robot d'exploration ");
    Scanner scanner = new Scanner(System.in);
    double energy = scanner.nextDouble();
    Robot robot = new Robot(energy, new Battery());
    boolean quitter = false;
    while (!quitter) {
        displayMenu();
        String commande;
        do {
            commande = scanner.nextLine();
        } while (commande.length() != 1);
        switch (commande.charAt(0)) {
            case 'A':
                System.out.println("coordonnées colonne,ligne de dépose du robot");
                Coordinates coord = lireCoordonnee(scanner);
                try {
                    robot.land(coord, new LandSensor(new Random(10)));
                } catch (UnlandedRobotException e) {
                    e.printStackTrace();  //To change body of catch statement use File | Settings | File Templates.
                } catch (LandSensorDefaillance landSensorDefaillance) {
                    landSensorDefaillance.printStackTrace();  //To change body of catch statement use File | Settings | File Templates.
                }
                break;
            case 'Z':
                try {
                    robot.moveForward();
                } catch (UnlandedRobotException e) {
                    System.out.println("Le robot est encore en l'air, il doit se poser d'abord");
                } catch (InsufficientChargeException e) {
                    System.out.println("Oups, piles vides... soyez patient, le soleil fait son oeuvre");
                } catch (LandSensorDefaillance landSensorDefaillance) {
                    System.out.println("Aie, le module de détection du terrain est défaillant. Abandon de l'exploration");
                    throw new RuntimeException(landSensorDefaillance);
                } catch (InaccessibleCoordinate inaccessibleCoordinate) {
                    System.out.println("le terrain devant le robot n'est pas praticable");
                }
                break;
            case 'S':
                try {
                    robot.moveBackward();
                } catch (UnlandedRobotException e) {
                    System.out.println("Le robot est encore en l'air, il doit se poser d'abord");
                } catch (InsufficientChargeException e) {
                    System.out.println("Oups, piles vides... soyez patient, le soleil fait son oeuvre");
                } catch (LandSensorDefaillance landSensorDefaillance) {
                    System.out.println("Aie, le module de détection du terrain est défaillant. Abandon de l'exploration");
                    throw new RuntimeException(landSensorDefaillance);
                } catch (InaccessibleCoordinate inaccessibleCoordinate) {
                    System.out.println("le terrain derrière le robot n'est pas praticable");
                }
                break;
            case 'Q':
                try {
                    robot.turnLeft();
                } catch (UnlandedRobotException e) {
                    System.out.println("Le robot est encore en l'air, il doit se poser d'abord");
                } catch (InsufficientChargeException e) {
                    System.out.println("Oups, piles vides... soyez patient, le soleil fait son oeuvre");
                }
                break;
            case 'D':
                try {
                    robot.turnRight();
                } catch (UnlandedRobotException e) {
                    System.out.println("Le robot est encore en l'air, il doit se poser d'abord");
                } catch (InsufficientChargeException e) {
                    System.out.println("Oups, piles vides... soyez patient, le soleil fait son oeuvre");
                }
                break;
            case 'L':
                try {
                    robot.cartographier();
                } catch (LandSensorDefaillance landSensorDefaillance) {
                    System.out.println("Impossible d'établir une cartographie");
                    break;
                } catch (UnlandedRobotException e) {
                    System.out.println("Le robot est encore en l'air, il doit se poser d'abord");
                }
                for (String ligne : robot.carte()) {
                    System.out.println(ligne);
                }
                break;
            case 'M':
                System.out.println("coordonnées colonne,ligne de la destination");
                Coordinates destination = lireCoordonnee(scanner);
                try {
                    robot.computeRoadTo(destination);
                } catch (UnlandedRobotException e) {
                    System.out.println("Impossible d'établir une route, le robot est encore en l'air");
                    break;
                } catch (LandSensorDefaillance landSensorDefaillance) {
                    System.out.println("Allo Houston, on a un problème. On a perdu le retour sol");
                    break;
                } catch (UndefinedRoadbookException e) {
                    System.out.println("Impossible d'établir une route, la destination est inatteignable");
                }
                try {
                    List<CheckPoint> checkPoints = robot.letsGo();
                    for (CheckPoint point : checkPoints) {
                        System.out.println("Position : (" + point.position.getX() + " ," + point.position.getY() + ") - orientation : " +
point.direction);
                    }
                } catch (UnlandedRobotException e) {
                    System.out.println("Le robot est encore en l'air, il doit se poser d'abord");
                } catch (UndefinedRoadbookException e) {
                    System.out.println("Il semblerait que le road book soit manquant");
                } catch (InsufficientChargeException e) {
                    System.out.println("Désolé, je n'ai pas suffisamment d'énergie pour terminer mon parcours");
                    displayBlackBox(robot.blackBox);
                } catch (LandSensorDefaillance landSensorDefaillance) {
                    System.out.println("Allo Houston, on a un problème. On a perdu le retour sol");
                    displayBlackBox(robot.blackBox);
                } catch (InaccessibleCoordinate inaccessibleCoordinate) {
                    System.out.println("Je suis malheureusement dans l'impossibilité de rejoindre la destination demandée");
                    displayBlackBox(robot.blackBox);
                }
                break;
            case 'X':
                quitter = true;
        }
    }
}
```