1. Part A and B
   1. getCubeTosses

| public static int[] getCubeTosses (NumberCube cube, int numTosses){  Array nums = {“1”, “2”, “3”, “4”, “5”, “6”};  Array<String> final = new Array<String>();  for(int i = 0; i <= numTosses; i++){  int rand = (int)(Math.random \* 6 + 1);  final.add(nums[rand]);  }  } |
| --- |

* 1. getLongestRun

| public static int getLongestRun(int [] values){  int index = 0;  for(int i = 0; i < values.length; i++){  String[] stringValues = Integer.parseInt(values[i]);  }  for(String value : stringValues){  for(int j = 1; j < stringValues.length; j++){  if(stringValues.length == 0){  index = -1;  }  else if(stringValues.length == 1){  index = Integer.parseInt(stringValues[1]);  }  else if(value.equals(stringValues[j]) && value.equals(stringValues[j - 1]) && value.equals(stringValues[j + 1])){  index = Integer.parseInt(value);  }  }  }  return index;  } |
| --- |

1. Part A and B
   1. getChargingCost

| private int getChargingCost(int starthour, int chargeTime){  int count = 0;  for(int i = startHour; i <= (startHour + chargeTime); i++){  count += rateTable[i % 24];  }  return count;  } |
| --- |

* 1. getChangeStartTime

| public int getChargeStartTime(int chargeTime){  int lowestCost = 9000;  int lowestHour = 24;  for(int i = 0; i < rateTable.length; i++){  int current = 0;  current = getChargingClass(i, chargeTime);  if(current < lowestCost){  lowestCost = current;  lowestHour = i;  }  }  return lowestHour;  } |
| --- |

1. Parts A and B
   1. getIndexForFit

| private int getIndexForFit(NumberTile tile){  int count = 0;  int index = 0;  if(board.size == 0){  index = -1;  }  else if(board.size == 1){  for(int j = 0; j <= board.size; j++){  if(count == 0){  if(board[j + 1].getLeft() == tile.getLeft()){  count++;  index = j;  }  }  }  }  else{  for(int i = 0; i <= board.size; i++){  if(count == 0){  if(board[i].getRight() == tile.getRight()){  count++;  }  if(count == 1){  if(board[i + 1].getLeft() == tile.getLeft()){  count++;  index = i;  }  }  }  }  if(count == 0){  index = -1;  return index;  } |
| --- |

* 1. insertTile

| public boolean insertTile(NumberTile tile){  boolean worked = false;  int rotationCount = 0;  int indexPossible = getIndexForFit(tile);  while(indexPossible == -1 && rotationCount < 4){  tile.rotate();  indexPossible = getIndexForFit(tile);  rotationCount++;  }  if(rotationCount >0){  board[indexPossible].add(tile);  if(board[indexPossible].getLeft() == tile.getLeft() && board[indexPossible].getRight() == tile.getRight){  worked = true;  }  else{  worked = false;  }  }  else{  worked = false;  }  return worked;  } |
| --- |