########################
GENERADOR DE KAKUROS:
########################
El generador de kakuros rebrà les dimensions x i y del kakuro a generar, el nombre de caselles blanques totals que contindrà i, per últim, quantes hauran de ser retornades amb el seu valor correcte. D'aquest mode, generarà un kakuro (TaulerEnunciat) amb una única solució amb totes les caselles negres amb els seus valors de Fila i Columna adients i les caselles blanques amb o sense valor, depenent de l'últim paràmetre que rep.

Per començar, l'algoritme genera una matriu de caselles i converteix totes les caselles de la columna 0 i fila 0 en caselles negres sense valor. Acte seguit, converteix la resta de caselles de la matriu en caselles blanques sense valor.

Ara, per tal d'evitar generar kakuros amb files o columnes amb més de 9 caselles blanques seguides, es recorre la matriu buscant més de 9 caselles blanques seguides, i quan trobem un cas, col·loquem una casella negra en una de les 3 caselles blanques anteriors recorregudes. Cada cop que es dóna aquesta circumstància, s'incrementa en 1 el valor de una variable que contarà quantes caselles negres hem col·locat en aquest recorregut per la matriu. Acte seguit, calculem el nombre de caselles negres restants per col·locar per tal que la matriu de caselles tingui el nombre demanat de caselles blanques i col·loquem les caselles negres restants.

Ara ja tenim la forma de la matriu de caselles amb cada casella sent del tipus que li toca (negra o blanca). Amb això fet, per cada casella negra que tingui una casella blanca (o més) a la dreta de la seva fila, contarem quantes blanques hi ha a la seva dreta i li assignarem a la casella negra un valor de suma de fila que tingui únicament 1 possible combinació de valors que la satisfaci (informació extreta de la classe Combinacions). Per exemple, per una casella negra amb dues caselles blanques a la dreta, el seu valor de suma de fila serà igual a 3 (1 + 2), a 4 (1 + 3), a 16 (7 + 9) o a 17 (8 + 9). D'aquesta forma, recorrerem totes les files emplenant totes les caselles negres que hagin de tenir valor de suma de fila un valor aleatori que només tingui 1 combinació possible.

Ara, amb aquesta matriu de caselles, crearem un TaulerComencat que cridarà a solucionaKakuro, una funció que farà funcions similars al solucionador de Kakuros explicat al següent apartat però que no mirarà que es compleixi la suma de valors a les columnes (doncs ara mateix totes les caselles negres tenen el valor de suma de columna = null). Dins aquesta funció, per cada casella blanca mitjançant el backtracking li assignarem a cadascuna tots els possibles valors que siguin: valors que pertanyen a la única combinació que fa que es satisfagui la suma de la fila (sense repetir cap valor òbviament) i valors que no apareguin a la mateixa columna que la casella blanca que estiguem tractant.

D'aquesta manera, solucionaKakuro ens retornarà un TaulerComencat que tindrà totes les caselles blanques amb valor si té solució. En cas de no tenir totes les caselles blanques amb valor (o de ser = null), retornarem una crida recursiva a generarKakuro per mirar de generar un nou Kakuro amb les mateixes dimensions de x, de y i amb el mateix numero de caselles blanques. Si totes les caselles blanques tenen valor, vol dir que ja tenim el kakuro que voliem generar i només ens quedaran un parell d'operacions per retornar el TaulerEnunciat.

Primer, recorrerem totes les columnes omplint les caselles negres amb la suma de columna que serà igual a la suma de tots els valors de les caselles blanques seguides que estiguin a sota de la casella negra tractada. Finalment, només ens caldrà treure el valor a caselles blanques (valor = null) de la matriu de caselles fins que només el nombre requerit de caselles blanques tinguin valor. Finalment, retornarem el TaulerEnunciat creat a partir d'aquesta matriu de caselles.

D'aquesta manera, hem creat un Kakuro amb solució única ja que no podem canviar el valor de cap casella blanca de la solució del Kakuro doncs violariem la suma de la Fila (una única combinació pel valor de la suma de la fila) i no podem permutar els valors de diferentes caselles blanques de la mateixa fila perquè també violariem la suma de les seves columnes.

Com a extra, també vam implementar un algorisme generador de kakuros que no dóna una solució única, per això ha quedat substituït per l'algorisme explicat anteriorment. Aquest segon generador encara hi és a la classe Algoritme sota el nom generarKakuroSimple, però no serà utilitzat en cap cas.

Estructures de dades utilitzades:

- Una matriu de caselles (Casella[][]) per a la realització d'algunes operacions com la assignació inicial de caselles blanques i negres.
- Un ArrayList<ArrayList<Integer> > (9) per emmagatzemar totes les possibles sumes amb combinació única per cada nombre de caselles blanques (tamany 9 perquè mai pot haver una fila amb més de 9 caselles blanques seguides ni menys de 1).
- Un TaulerComencat tc (creat a partir de la matriu de caselles comentada anteriorment) que utilitzarem per a les crides a les funcions auxiliars comentades anteriorment(per exemple solucionaKakuro).
- Un TaulerEnunciat creat a partir de la matriu de caselles anteriorment comentada quan aquesta ja té el nombre adequat de caselles blanques amb valor i sense (i per tant també una solució única i vàlida).

A les funcions auxiliars utilitzarem les següents estructures de dades:

- Un Set<Integer> que s'encarregarà de guardar els possibles valors que pot tenir una casella en funció de la única combinació possible de valors de la seva fila i dels valors de la esmentada combinació que ja han estat utilitzats en caselles blanques anteriors.
- Un altre Set<Integer> que emmagatzemarà els valors (sempre entre 1 i 9) que altres caselles blanques ja tenen a la columna de la casella que s'estigui tractant.

##########
SOLUCIONADOR DE KAKUROS:
##########
El solucionador de kakuros rebrà un (TaulerEnunciat) vàlida, generarà un munt de combinacions diferents fins que, amb ajuda del algoritme de validador de solució trobi la

primera que compleix els requisits d'un Kakuro resolt i, en cas d'haver trobat una solució, retornarà un (TaulerComencat) amb totes les caselles blanques omplides o en cas contrari de no haver trobat cap solució, retornarà un null que representarà com si d'aquell enunciat donat, no hi existeix cap solució.

Com que hem investigat i trobat que la quantitat de combinacions possibles són bastant limitats, hem decidit deixar totes les possibles combinacions hardcodejats dins de la classe Combinacions amb els seus corresponents getters que retornen una serie de combinacions desitjats, per tant, la lògica del nostre solver depèn bàsicament la classe Combinacions, on donat un nombre de valors a sumar(número de caselles blanques) i la seva suma (número que han de sumar les caselles blanques que apareix a la seva respectiva casella negra), és capaç de generar una llista de possibles valors per provar de manera iteratiu per cada casella, ja que es pot aplicar la mateixa fórmula de : (valor fila/columna casella negra - suma de caselles blanques ja col·locades) i (número de caselles totals - número de caselles blanques ja col·locades) per a cada una de les caselles blanques.

El nostres solver està format per un conjunt de senzills mètodes privats que sumen, recorren i busquen caselles dins d'un tauler. D'aquesta manera hem pogut programar un mètode privat recursiva de resoldreKakuro que cridat pel seu mètode públic auxiliar de resoldreKakuro que prova el validador de solucions amb totes les possibles solucions.

Al recorrer el kakuro amb un backtracking i de manera recursiva, el nostre programa ja és capaç de trobar la solució, tot i així té una gran limitació, ja que al existir tantes possibles combinacions juntament amb una complexitat de n factorial, el nostre algoritme no pot resoldre kakuros més enllà del 5*5. Per tant hem decidit utilitzar els Set<Integer> que al ser de tipus HashSet, no deixa repetir valors dins d'un mateix set. D'aquesta manera i aprofitant que es pot aplicar interseccions de tres sets per trobar els valors que no es repeteixen ni apareixen entre les combinacions possibles d'una casella blanca amb la seva corresponent fila i columna de valors ja col·locats. D'aquesta manera hem aconseguit que el nostre algoritme pugui reduir notòriament la quantitat de combinacions possibles i reduir la seva complexitat fins a resoldre kakuros de 20*20! (Comprovat desde un ordinador personal de torre).

Estructures de dades utilitzades:

- TaulerComencat i TaulerEnunciat: que està format per Casella[][] per poder llegir, escriure i modificar les coordenades i els valors de totes les caselles tan blanques com negres.
- HashSet<Integer>: Per a generar les possibles combinacions a probar de cada CasellaBlanca que s'han obtingut de la classe Combinacions i per aplicar interseccions i reduir els casos possibles.

##########
VALIDADOR DE FORMAT:
##########
El validador de Format de kakuros rep un objecte de la classe TaulerEnunciat i comprova que el format del tauler sigui correcte. Per format s'entenen coses bàsiques com:

- La fila i columna “0” han de ser negres.
- Que les caselles de la fila inicial no tinguin valor de Fila
- Que les caselles de la columna inicial no tinguin valor de Columna
- Que la casella (0,0) no tingui valor de Fila ni de Columna
- Que les caselles negres tinguin un valor de fila i columna possible donat el número de caselles blanques i el seu valor si en tenen.
  Això es calcula de la següent forma:
  (N*N + N)/2 + acum <= valor <= 9*N - (N*N - N)/2 + acum ; on
  N: és el número de caselles blanques
  acum: és la suma dels valors ja establerts
  valor: és el valor de la fila o columna corresponent
- Que les caselles negres tinguin valor de fila i columna si i només si n’han de tenir.

És un algoritme senzill que simplement fa diverses iteracions sobre el tauler, tot comprovant aquestes coses, i que avisa amb l’error concret que ha detectat.
En concret, l’algoritme recorre les columnes i files del final cap al principi, i va acumulant els valors i va contant les caselles blanques, i quan es troba amb una casella negra fa les comprovacions, i llavors segueix fins arribar al principi.

##########
VALIDADOR DE SOLUCIÓ:
##########
El validador de Solucions de kakuros rep un objecte de la classe TaulerComencat amb un tauler tot emplenat i comprova que la proposta de solució és correcte.
Fa les següents comprovacions:
- Que totes les caselles blanques tenen un valor
- Que totes les files i columnes sumen lo que han de sumar

L’algoritme en sí és molt senzill, fa una sola iteració sobre el tauler fent aquestes comprovacions per fila i per columna.