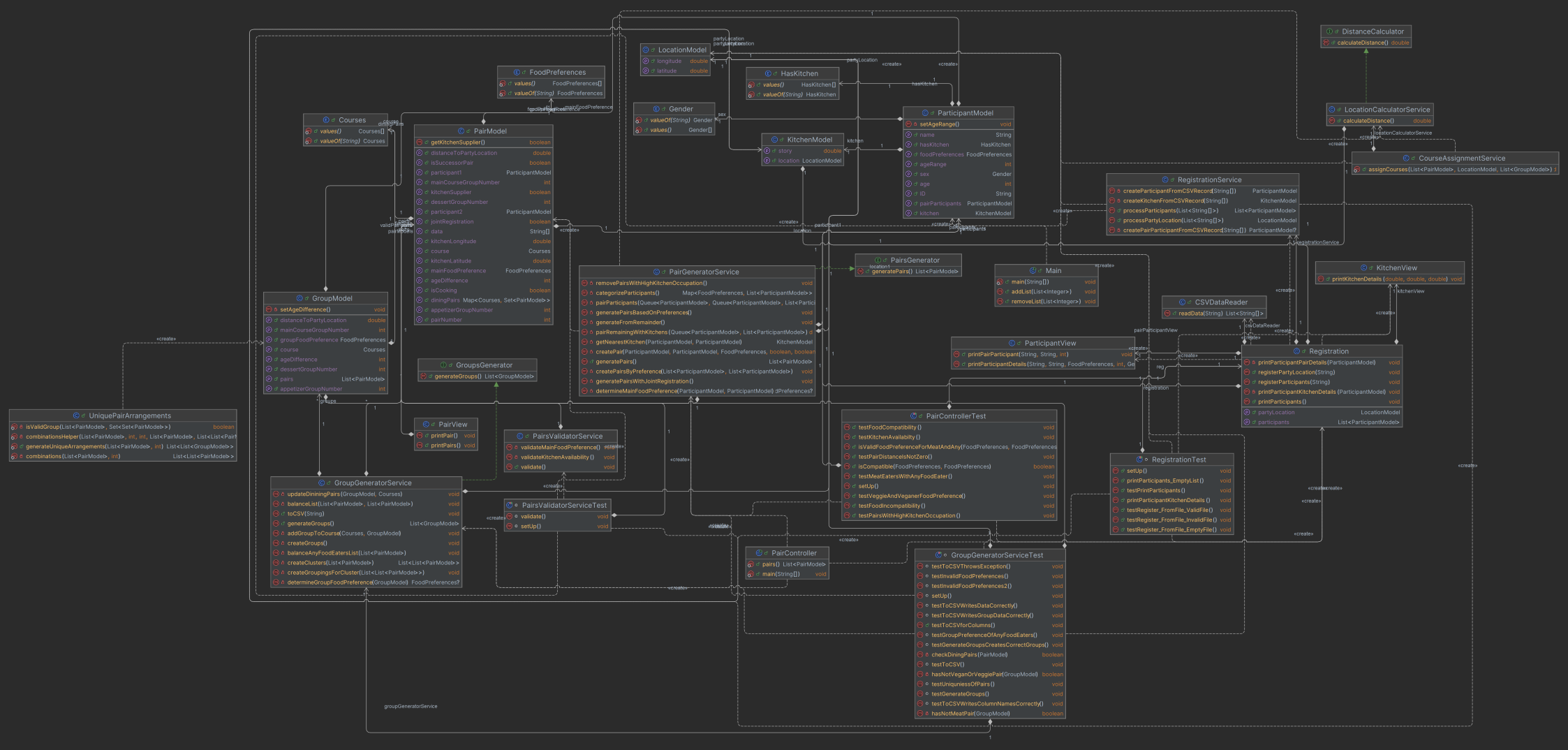
LocationModel
longitude double
latitude double
HasKitchen
values() HasKitchen[]
valueOf(String) HasKitchen
Gender
valueOf(String) Gender
values() Gender[]
KitchenModel
story double
location LocationModel
ParticipantModel
setAgeRange() void
name String
hasKitchen HasKitchen
foodPreferences FoodPreferences
ageRange int
sex Gender
age int
ID String
pairParticipants ParticipantModel
kitchen KitchenModel
DistanceCalculator
calculateDistance() double
LocationCalculatorService
calculateDistance() double
CourseAssignmentService
assignCourses(List<ParticipantModel>, LocationModel, List<GroupModel>)
FoodPreferences
values() FoodPreferences[]
valueOf(String) FoodPreferences
Courses
values() Courses[]
valueOf(String) Courses
PairModel
getKitchenSupplier() boolean
distanceToPartyLocation double
isSuccessorPair boolean
participant1 ParticipantModel
mainCourseGroupNumber int
kitchenSupplier boolean
dessertGroupNumber int
participant2 ParticipantModel
jointRegistration boolean
data String[]
kitchenLongitude double
course Courses
kitchenLatitude double
mainFoodPreference FoodPreferences
ageDifference int
isCooking boolean
diningPairs Map<Courses, Set<PairModel>>
appetizerGroupNumber int
pairNumber int
RegistrationService
createParticipantFromCSVRecord(String[]) ParticipantModel
createKitchenFromCSVRecord(String[]) KitchenModel
processParticipants(List<String[]>) List<ParticipantModel>
processPartyLocation(List<String[]>) LocationModel
createPairParticipantFromCSVRecord(String[]) ParticipantModel?
PairsGenerator
generatePairs() List<PairModel>
Main
main(String[])
addList(List<Integer>) void
removeList(List<Integer>) void
KitchenView
printKitchenDetails(double, double, double) void
GroupModel
setAgeDifference() void
distanceToPartyLocation double
mainCourseGroupNumber int
groupFoodPreference FoodPreferences
course Courses
ageDifference int
dessertGroupNumber int
pairs List<PairModel>
appetizerGroupNumber int
PairGeneratorService
removePairsWithHighKitchenOccupation()
categorizeParticipants() Map<FoodPreferences, List<ParticipantModel>>
pairParticipants(Queue<ParticipantModel>, Queue<ParticipantModel>, List<Partic
generatePairsBasedOnPreferences()
generateFromRemainder()
pairRemainingWithKitchens(Queue<ParticipantModel>, List<ParticipantModel>)
getNearestKitchen(ParticipantModel, ParticipantModel) KitchenModel
createPair(ParticipantModel, ParticipantModel, FoodPreferences, boolean, boolean
generatePairs() List<PairModel>
createPairsByPreference(List<ParticipantModel>, List<ParticipantModel>)
generatePairsWithJointRegistration()
determineMainFoodPreference(ParticipantModel, ParticipantModel) dPreferences?
CSVDataReader
readData(String) List<String[]>
ParticipantView
printPairParticipant(String, String, int) void
printParticipantDetails(String, String, FoodPreferences, int, Ge
Registration
printParticipantPairDetails(ParticipantModel) void
registerPartyLocation(String) void
registerParticipants(String) void
printParticipantKitchenDetails(ParticipantModel) void
printParticipants() void
partyLocation LocationModel
participants List<ParticipantModel>
GroupsGenerator
generateGroups() List<GroupModel>
UniquePairArrangements
isValidGroup(List<PairModel>, Set<Set<PairModel>>) boolean
combinationsHelper(List<PairModel>, int, int, List<PairModel>, List<List<Pair
generateUniqueArrangements(List<PairModel>, int) List<List<GroupModel>>
combinations(List<PairModel>, int) List<List<PairModel>>
PairView
printPair() void
printPairs() void
PairsValidatorService
validateMainFoodPreference() void
validateKitchenAvailability() void
validate() void
PairControllerTest
testFoodCompatibility() void
testKitchenAvailability() void
isValidFoodPreferenceForMeatAndAny(FoodPreferences, FoodPreferences
testPairDistanceIsNotZero() void
isCompatible(FoodPreferences, FoodPreferences) boolean
testMeatEatersWithAnyFoodEater() void
setUp() void
testVeggieAndVeganerFoodPreference() void
testFoodIncompatibility() void
testPairsWithHighKitchenOccupation() void
RegistrationTest
setUp() void
printParticipants_EmptyList() void
testPrintParticipants() void
printParticipantKitchenDetails() void
testRegister_FromFile_ValidFile() void
testRegister_FromFile_InvalidFile() void
testRegister_FromFile_EmptyFile() void
GroupGeneratorService
updateDiningPairs(GroupModel, Courses) void
balanceList(List<PairModel>, List<PairModel>) void
toCSV(String) void
generateGroups() List<GroupModel>
addGroupToCourse(Courses, GroupModel) void
createGroups() void
balanceAnyFoodEatersList(List<PairModel>) void
createClusters(List<PairModel>) List<List<PairModel>>
createGroupingsForCluster(List<List<PairModel>>) void
determineGroupFoodPreference(GroupModel) FoodPreferences?
PairsValidatorServiceTest
validate() void
setUp() void
PairController
pairs() List<PairModel>
main(String[]) void
GroupGeneratorServiceTest
testToCSVThrowsException() void
testInvalidFoodPreferences() void
testInvalidFoodPreferences2() void
setUp() void
testToCSVWritesDataCorrectly() void
testToCSVWritesGroupDataCorrectly() void
testToCSVforColumns() void
testGroupPreferenceOfAnyFoodEaters() void
testGenerateGroupsCreatesCorrectGroups() void
checkDiningPairs(PairModel) boolean
testToCSV() void
hasNotVeganOrVeggiePair(GroupModel) boolean
testUniquenessOfPairs() void
testGenerateGroups() void
testToCSVWritesColumnNamesCorrectly() void
hasNotMeatPair(GroupModel) boolean
groupGeneratorService