

**Developer**: Jermaine Wiggins

**Date**: June 2, 2024

# IT 145 Global Rain Summary Report Template

## Pseudocode

**P****et Check out method**

**Variables**

Cats array list

Dogs array list

daysStay = length of stay

catSapceNumber = number assign to cat , array index

dogSpaceNumber = number assigned to dog, array index

catSpaces = 12

dogSpaces = 30

**START** pickup

**SET** amount due to 0

**PROMPT** user for pet name and type (dog/cat)

**IF** pet is cat

**GET** catSpaceNumber

**FIND** pet in Cats array

**GET** daysStay

**CALCULATE** amount due

**REPLACE** cat name with vacant in Cat array list

**ADD** one to catSpaces

**ELIF** pet is dog

**GET** dogSpaceNumber

**FIND** pet in Dog array

**GET** dogWeight

**GET** fee based on dogWeight

**GET** daysStay

**IF** daysStay >= 2 and getGrooming == True

**ADD** grooming fee(based on dogWeight) to amount due

**CALCULATE** amount due

**REPLACE** dog name with vacant in Dogs array list

**ADD** one to dogSpaces

**ELSE**

**PRINT** Not a valid pet name or type

**ENDIF**

**PRINT** amount due

**PRINT** pet name has checked out

## Flowchart

**A diagram of a cat

Description automatically generated**

## OOP Principles Explanation

The OOP principles help create better designed code that is easier to read and troubleshoot. I have implemented these principles in my software development by defining methods. Methods allow me to hide the complex parts of the code and instead use names that describe what the methods do, improving readability and thus applying the abstraction principle. Instead of showing how I would retrieve the number of days stayed, I can simply call the method. Another way I have implemented these principles is through encapsulation, which involves hiding private data, like int dog space number within a class, and allowing access only through mutators and accessors  (setDogSpaceNumber and getDogSpaceNuumber). Finally, I have incorporated inheritance and polymorphism by creating parent (Pet) and child classes (Cat and Dog) that can call methods that the parent class has, such as getDaysStay, and use methods in their unique ways compared to another child class, such as getAmoutDue.