**Data Structures and Object Oriented Fundamentals**

Data types are broken down into two major types: Primitive and Non-Primitive. Below are two charts showing the breakdown of these types into their subtypes.

| **Non-Primitive** |
| --- |
| Strings |
| Arrays |
| etc. |

| **Primitive** | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Boolean** | **Numeric** | | | | | | |
| **||** | **Character** | **Integral** | | | | | |
| **||** | **||** | **Integer** | | | | **Floating-point** | |
| boolean | char | byte | short | int | long | float | double |

Now to breakdown the integral values by their ranges and storage size. This allows you to know how much memory is reserved for these data types

| **Integers** | | |
| --- | --- | --- |
| **Type:** | **Range** | **Storage Size** |
| **byte** | -128 to 127 | 8-bit signed |
| **short** | -32768 to 32767 | 16-bit signed |
| **int** | -2147483648 to 2147483647 | 32-bit signed |
| **long** | -9223372036854775808 to 9223372036854775807 | 64-bit signed |

| **Floating-point** | | |
| --- | --- | --- |
| **Type:** | **Range** | **Storage Size** |
| **float** | Negative Range: -3.4028235E+38 to -1.4E-45  Positive Range: 1.4E-45 to 3.4028235E+38 | 32-bit IEEE 754 |
| **double** | Negative Range: -1.7976931348623157E+308 to -4.9E-324  Positive Range:4.9E-324 to 1.7976931348623157E+308 | 32-bit IEEE 754 |

It is important to note that you can use a larger data type to hold smaller values but not the other way around. Along with not being able to store a value assigned as another type to one of another type, without casting.

Casting is when you convert one data type into another data type and can be done implicitly or explicitly. An example of an implicit cast is double dollars = 20; and an example on explicit is int countLimit = 4.0;.

Along with these variable types there are variables that are called instance variables. These variables are a part of specific methods instead of the whole class itself. The methods these variables are in are called instance methods which are called by reference to an instance within a class, usually a method associated with an object. There is also static variables which are variables that are assigned to the class as a whole rather than a specific instance.

Now for Classes and Objects to be discussed. A class is seen like a blueprint for your code that tells you how to make other classes like it. While an Object is what that blueprint has created. When creating an object you might want to start with a UML(unified Modeling Language) diagram to help you figure out what is needed in your Object. Below is an example of a UML.

| Animal | Class name |
| --- | --- |
| Name  Food  Weight  Sleep  Location | Data Fields |
| Animal()  getName()  getFood()  getWeight()  getSleep()  getLocation()  eat()  sleep()  swim() | Constructor and methods |

The UML helps with constructors as well which are methods within a class that describe the object of the class and share the name with the class itself. In some cases you might want to make a more specific class based off another class and that is what we call subclasses. These subclass will take portions from the parent class and override other portions to display different information. Like if we have a parent class that has outputs for sleeping, eatting, and swimming but want the subclass to change these values like so.

class Animal

public void eat()

{

System.***out***.println("Animal is eatting");

}

public void sleep()

{

System.***out***.println("Animal is sleeping");

}

public void swim()

{

System.***out***.println("Animal is swimming");

}

class Bear extends Animal

*@Override*

public void eat()

{

System.***out***.println("Bear is eatting some " + this.getFood());

}

*@Override*

public void sleep()

{

System.***out***.println("Bear is asleep -KEEP QUIET-");

System.***out***.println("sleeping for the next " + this.getSleep() + " Hrs");

}

*@Override*

public void swim()

{

System.***out***.println("Bear is swimming around");

}

Now for us to discuss how memory is stored. There are two zones where memory is stored. The first one is the stack which copies the initial state of what is stored there and any changes to the item stored doesn’t affect the currently stored value. The other is heap which does alter the stored value as changes are made.

Sometimes you don’t want to see everything that the code is doing, like all of the variables. This is where Encapsulation comes in. Encapsulation is the combination of variables and methods within a class that allows implementation without showing every little detail.

**Strings, Arrays and Array of Objects**

Strings are like lines of a book in the fact that they usually contain letters(chars) and are used to display text to the user by using System.out.print in its many forms. Strings can be used to hold names, titles, and whole sections of text if needed.String charName = "Simon";

Arrays are another Non-Primitive data type like Strings. Arrays hold groups of information in indexes within the array for reference and use later. Arrays do have limits though, which are set at creation known as the array size. Arrays indexes start at 0 and go up until right before their size number as seen below.

int[] numbers = new int[6];

| Index 0 | Index 1 | Index 2 | Index 3 | Index 4 | Index 5 |
| --- | --- | --- | --- | --- | --- |
| 70 | -23 | 14 | 8 | 33 | 27 |

Though arrays, as seen above, can be set up as empty arrays with a set size, you could also set up the array with the data already entered like: int[] numbers = {70,-23,14,8,33,27};. You can also make an array of objects which would use the name of your object as the variable type and you would assign each index a variable. Also each index of an array can be called like so: System.***out***.println("Animals[1] is a " + Animals[1]);.

**Inheritance and Polymorphism**

Inheritance is what we refer to when using parent(super) and child(subclass) classes. Inheritance is what a child class gains from their parent class that is not overridden. On the other hand Polymorphism is the portions from the parent class that the child class changes to suit their needs. Below is an example of results that come from inheritance and from polymorphism.

Inheritance from Animal Class

Animals[3] is a Sloth

Sloth Name: Sid - Weighs: 15 lbs - Sleeps: 12 hrs - Location: Urban Jungle

Animal is eatting

Animal is sleeping

Animal is swimming

Polymorphism from Animal Class

Animals[1] is a Bear

Bear Name: Po - Weighs: 550 lbs - Sleeps: 20 hrs - Location: Asian Passage

Bear is eatting some Bamboo

Bear is asleep -KEEP QUIET-

sleeping for the next 20 Hrs

Bear is swimming around

Within these two concepts comes a term known as Dynamic Binding. Dynamic Binding is when an output is determined during the runtime of the program rather than prior to the program. This is what decides what will be inherited from a parent class and what will end up being changed by the child class.

**Files and Exception Handling**

Programs have ways of reading and writing files. In order to let your program read the file you will need to call for the file File input = new File("Animal.txt");. To use this information from the file you will need to set up a scanner Scanner file = new Scanner (input);. With these things in place you can take information stored within that file to be used by your own code. In order to write/ create a file you will need to call for a new file(File out = new File(“nameWhatever”);) and apply that to PrintWriter(PrintWriter output = new PrintWriter(out);). As soon as you are done with the PrintWriter and Scanner you will need to use their variable\_name.close() in order to finalize them and not have your program waste more resources.

With these files and Scanner inputs coming into play a new problem arises from possible invalid information being entered. That is where Exception Handling comes in to help you out. Exception Handling allows you code to deal with errors without causing a whole crash to your program. These can be invalid data type exceptions or even missing file exceptions. With these safety nets you are able to bypass problems that you would have run into without them. The most common one is the Try-Catch method. With the Try-Catch method your code will go into the try portion of code and if there is a problem stops what it is doing and skips down to the catch section of code usually letting the user know that something was entered in correctly. Another is a throw statement that would be applied to the class itself rather than somewhere within the class public static void main(String[] args) throws FileNotFoundException.