**Amazing Adventure**

**inputStream** and **outputStream** for testing:

* I may have exaggerated the importance of using this approach. If you will recall, this approach allows you to compare strings produced in your methods to be compared to expected output in your tests.
* There WILL be a second week to Amazing Adventure.

**Abstract Classes and Interfaces**

**Abstraction** − Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.  
Since all the methods of the interface are abstract the user doesn’t know how a method is written (except the method signature/prototype). Using interfaces, you can achieve (complete) abstraction.

**Loose coupling** − Coupling refers to the dependency of one object type on another, if two objects are completely independent of each other and changes done in one don’t affect the other, both are said to be loosely coupled with respect to one another.

**Abstract Classes**

<https://beginnersbook.com/2013/05/java-abstract-class-method/>

* The purpose of an abstract class is to function as a base for subclasses.
* You can not declare an instance of an abstract class.
* An abstract class has no use until/unless it is extended by some other class.
* An abstract class can have:
  + Concrete methods
  + Abstract methods
* Every abstract method in the abstract class must be implemented in the classes that extend the class.
* You can not have an abstract method in a concrete class.
* An abstract class MAY have ONLY concrete methods (why would you do that? So you can OPTIONALLY override them)

**To Declare an Abstract Class:**

public abstract class MyAbstractClass {

}

**Declare an Instance of the class (this doesn’t work):**

MyAbstractClass myClassInstance = new MyAbstractClass(); //not valid

**How You Use an Abstract Class (Pet):**

**In my example,**

* “Pet” is the abstract class which declares both
  + concrete and
  + abstract methods.
* We declare “disp()” which is concrete
* we declare an abstract class that has a public method called “sound()”. we leave it undefined so that it is defined when we “extend” the parent class.
* classes that extend the parent class can contain methods not declared in the parent

**Java Interface (Animal)**

<https://www.tutorialspoint.com/what-is-the-purpose-of-interfaces-in-java>

* An interface in Java is a specification of method prototypes.
* When you need to guide the programmer, you can make a contract specifying what methods and fields of a type should be
* To create an object of this type you need to implement this interface, provide the body for all of the abstract methods of the interface and obtain the object of the implementing class.
* An interface looks like a class but it is not a class.
* Methods are
  + public
  + Abstract (contain no implementation)
* Variables are
  + public by default
  + static
  + final

**Static:**

* When a variable is declared as static, a single copy of the variable is created and shared among all objects at class level.
* Static variables are, essentially, global (to the class) variables. That is, all instances of the class share the same static variable.
* Example: int totalCopiesInUse;

**Final**: <https://www.geeksforgeeks.org/final-keyword-java/>

When referring to variables, “final” essentially create a constant.

When a variable is declared with *final* keyword, its value can’t be modified, essentially, a constant. This also means that you must initialize a final variable. If the final variable is a reference, this means that the variable cannot be re-bound to reference another object, but internal state of the object pointed by that reference variable can be changed i.e. you can add or remove elements from [final array](https://www.geeksforgeeks.org/final-arrays-in-java/) or final collection. It is good practice to represent final variables in all uppercase, using underscore to separate words.

**Interface vs Abstract Class**

<https://www.guru99.com/interface-vs-abstract-class-java.html>

**Interfaces:**

* The interface is a blueprint that can be used to implement a class. The interface does not contain any concrete methods (methods that have code).
* All the methods of an interface are abstract methods.
* Interfaces never contain instance variables but, they can contain public static final variables
* Can contain other interfaces

**Abstract Classes:**

* Abstract classes should have at least one abstract method. , i.e., methods without a body. It can have multiple concrete methods.
* Abstract classes allow you to create blueprints for concrete classes. But the inheriting class should implement the abstract method.

**\*Neither abstract classes nor interfaces can be instantiated.**

<https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/>

**When to Use:**

<https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html>

**Easiest Explanation:**

<https://medium.com/@alifabdullah/easiest-explanation-of-abstract-class-and-interface-280741bc2daf>

**What Happens When an Abstract Class Implements an Interface?**

<https://stackoverflow.com/questions/49757423/what-happens-when-an-abstract-class-implements-an-interface-in-java>

**Nested Interfaces:**

<https://beginnersbook.com/2016/03/nested-or-inner-interfaces-in-java/>

**Example uses:**

Game example uses:

* Chess pieces
* Land Vehicles
* Non-player characters
* Cards (foreshadowing)

Cars in a train:

* boxCar
* flatCar
* tankCar

Force all train cars to

* allow partial unloading of freight
* produce “volume” or “capacity” calculations