INTERFACES, ENUMS and User input

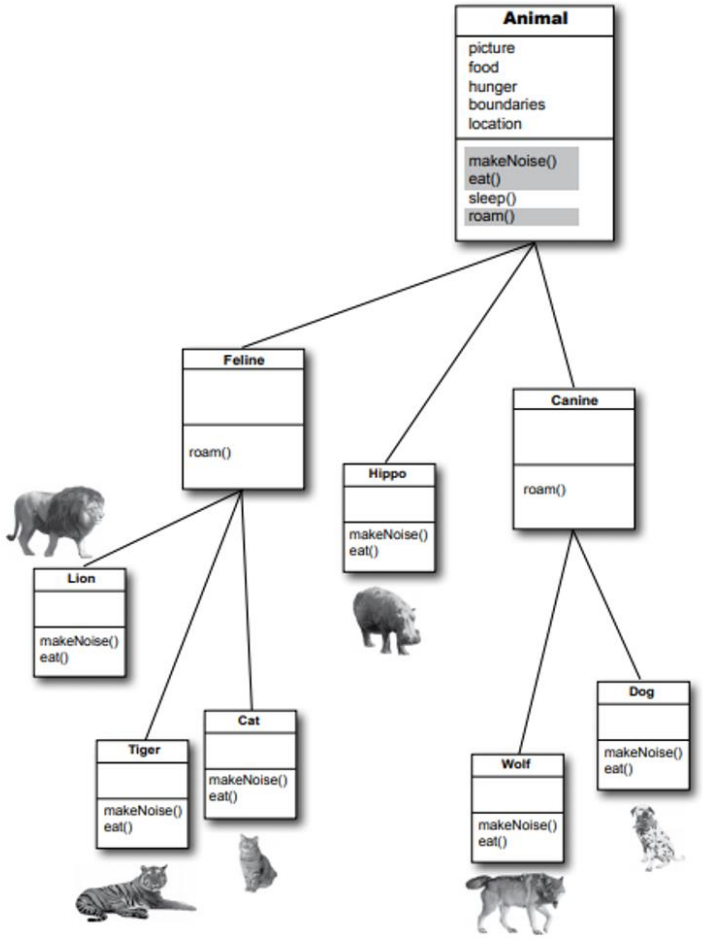
INTERFACES

INTERFACES

By the end of the discussion on interface, you will be able to:

- Define and use interface
- Determine the difference between abstract and interface
- Things to remember when using interface
- Decide between abstract classes and interface when designing class hierarchy

Let’s make an Animal Class!

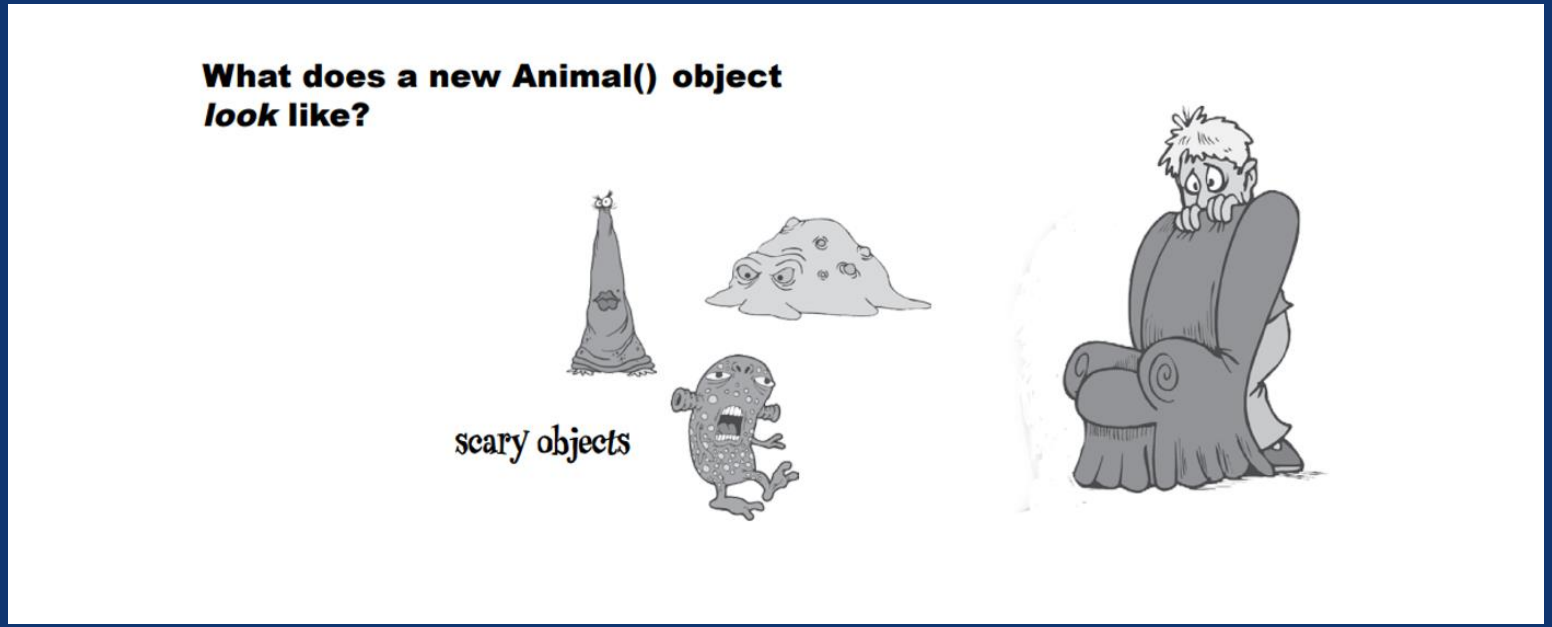

Animal
picture
food
hunger
boundaries
location
makeNoise()
eat()
sleep()
roam()
Feline
roam()
Hippo
makeNoise()
eat()
Canine
roam()
Lion
makeNoise()
eat()
Tiger
makeNoise()
eat()
Cat
makeNoise()
eat()
Wolf
makeNoise()
eat()
Dog
makeNoise()
eat()

We can say:

- Wolf aWolf = new Wolf();
- Animal aHippo = new Hippo();

- But here is where it gets weird:
- Animal anim = new Animal();
- Animal reference to an object and these two are the same types.
- But what the heck does an Animal object looks like??

Some classes just should not be Instantiated! Using abstract class will solve this problem.



How do we:

1. Force subclasses to have a method
2. Stop having actual objects of that class
3. Keep having references of that class
4. Retain common code of that class

These 4 can be fulfilled through the use of **abstract class and abstract method, however…**

WE WOULD WANT TO USE INTERFACE IF WE WANT TO

1. Force subclasses to have a method
2. Stop having actual objects of a class
3. Keep having references of that class

~~4. Retain common code of that class~~ - Not including this!

The methods inside the interface are by default, public, void (can be changed to a return type), and abstract. The variables are by default public, static, and final.

Common Code/
Defined Methods
Is possible for abstract classes

Java - ReportingInterfaces/src/test/Kane.java - Eclip

File Edit Source Refactor Navigate Search

Quick Access | Java

*Wrestler.java

```java
package test;

public abstract class Wrestler {

    public void paymentForWork (int hours) {
        System.out.println
        ("The wrestler will make $"+
        hours*250.00);
    }
    public abstract void themeMusic();
    public abstract void finisher();
}
```

Its also possible to have non-final, non-static, any access modifiers to variables and methods (except abstract methods)

```java
package test;

public class Kane extends Wrestler{

    @Override
    public void themeMusic() {
        System.out.println
        ("Kane's Intro music...");
    }
    @Override
    public void finisher() {
        System.out.println
        ("Kane's finishing TombStone...");
    }
}
```

Main.java

```java
package test;

public class Main {
    public static void main (String [] args) {
        Wrestler wrestler1= new Kane();

        wrestler1.themeMusic();
        wrestler1.paymentForWork(5);
        wrestler1.finisher();
    }
}
```

Writable | Smart Insert | 16 : 1

Type here to search

10:28 PM
6/8/2021

Java - ReportingInterfaces/src/test/

File Edit Source Refactor Navi

The methods created in interface must all be abstract and public. But it has exceptions.

Interfaces can also allow default(defined methods) and static methods, however it is situational.

*Wrestler.java

```java
package test;

public interface Wrestler {

    public  abstract void paymentForWork
        (int hours);

    public abstract void themeMusic();
    public abstract void finisher();
}
```

```java
package test;

public class StoneCold implements Wrestler {
    @Override
    public void paymentForWork(int hours) {
        System.out.println
            ("Stone cold will make $" +
            hours*300.00);
    }
    @Override
    public void themeMusic() {
        System.out.println
        ("Stone cold's music playing...");
    }

    @Override
    public void finisher() {
        System.out.println
        ("Stone cold's finishing stunner...");
    }
}
```

```java
package test;

public class Main {
    public static void main (String [] args) {
        Wrestler wrestler2= new StoneCold();

        wrestler2.themeMusic();
        wrestler2.paymentForWork(5);
        wrestler2.finisher();
    }
}
```

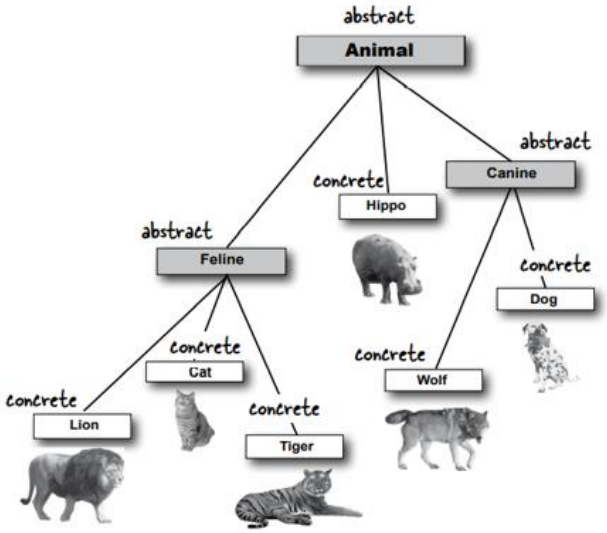
It uses the keyword “Implements”, not “extends”

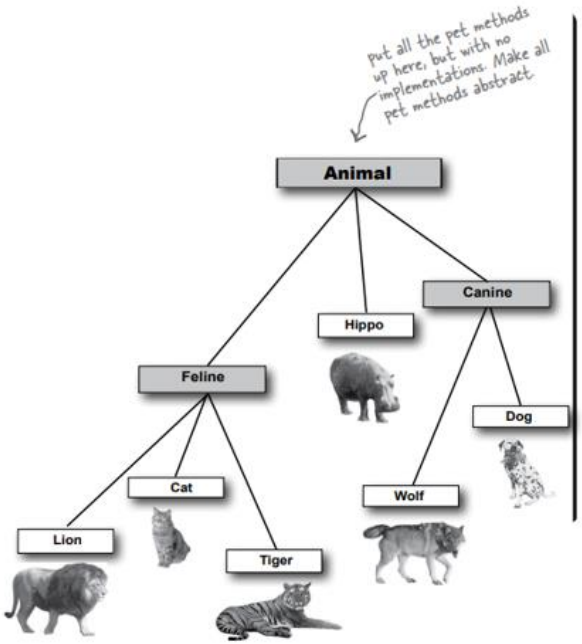
Writable | Smart Insert | 11 : 2

Type here to search

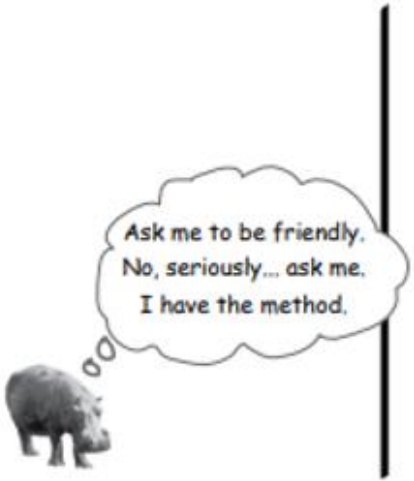
10:32 PM 6/8/2021

Let’s say that we are making a pet program! So where do we put the “friendly” methods?



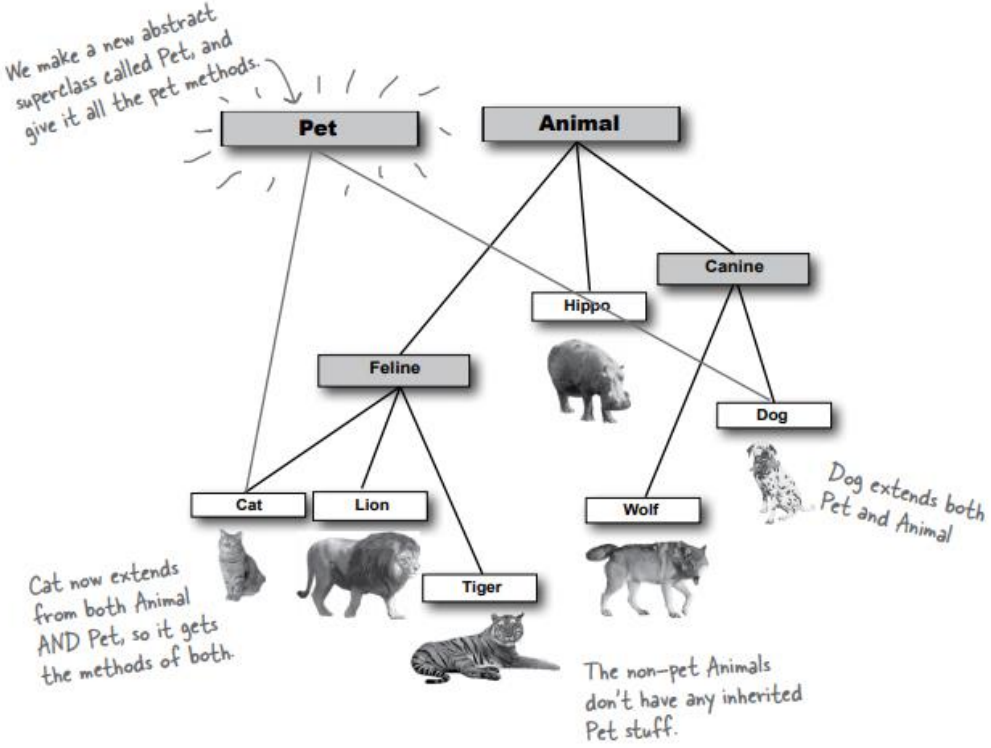


But something’s wrong!

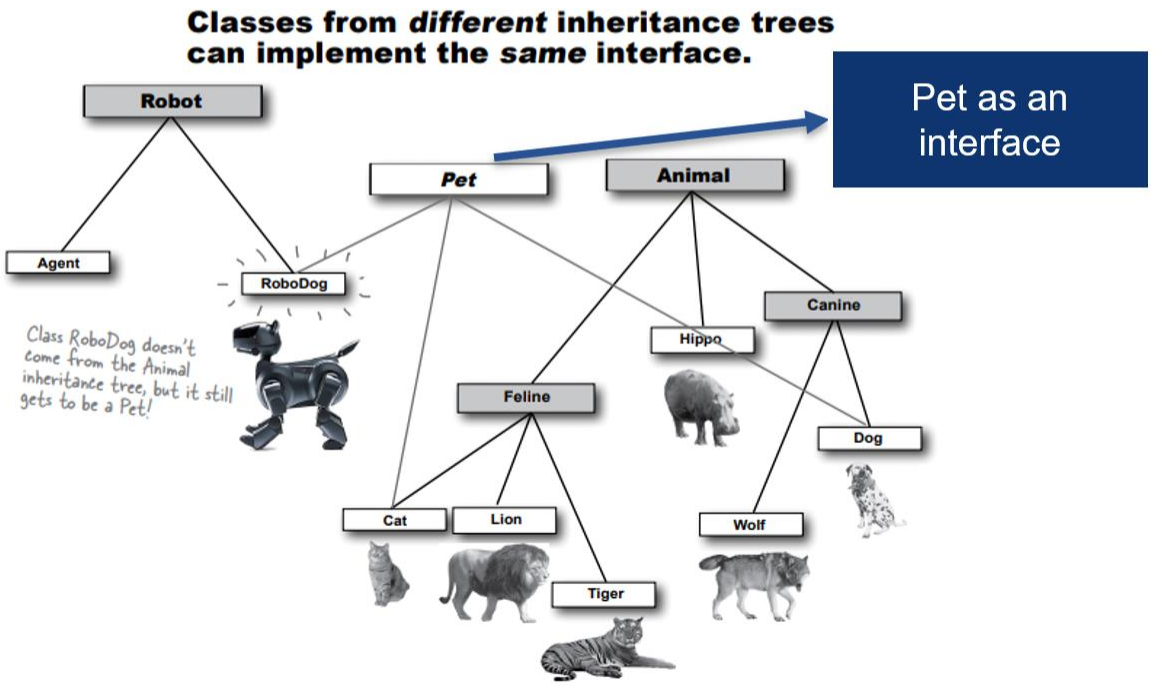


Maybe we can put the pet methods in the subclasses individually? But it would take too much time and the code isn’t neat anymore.

It looks like we need TWO superclasses at the top



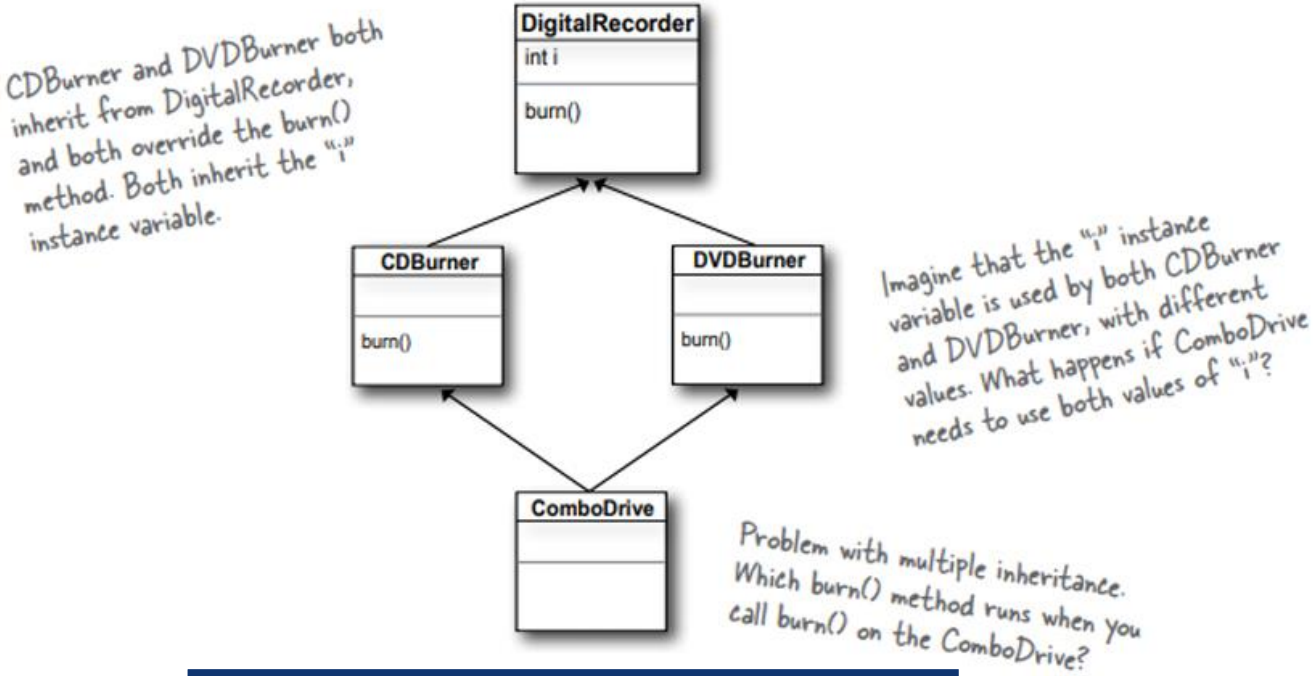
But in java you can only extend 1 class, That's where interface comes in.



Interface is:

- Not a GUI interface, but the Java keyword interface.
- When a class implements an interface, you can think of a class signing a contract, agreeing to perform specific behaviors of the interface. (Similar to inheriting methods, but the behavior must be defined in the class that implements interface).
- A Java interface solves your multiple inheritance problem by giving you much of the polymorphic benefits of multiple inheritance without the pain and suffering from the Deadly Diamond of Death (DDD).

Deadly Diamond of Death




```
ComboDrive x = new ComboDrive();
x.burn(); //which burn?
```

If you create a method inside an interface, it should be abstract, but…

- Generally, interface is an abstract class without any code.
- In Java 1, they added the keyword “interface” to apply this rule.
- Reengineering an existing JDK framework is always very complex. Modifying one interface in a JDK framework breaks all classes that extend the interface, which means that adding any new method could break millions of lines of code. Therefore, ***default methods*** have been introduced as a mechanism to extend interfaces in a backward-compatible way. Java 8 has introduced default methods.
- You could say that abstract class and interface are almost similar with the exception of some restrictions.
- Abstract class can define constructors, any access modifiers (except for abstract methods), and non-static, non-final member variables. The interface can’t do these.

IN GENERAL:

- If you just want to define a required method – then use Interface!
- If you want to define potentially required methods and common behavior – use Abstract class/method!

REMEMBER:

- Interface is not a class, it's just interface!
- A typical class "implements" the interfaces, not "extends"!
- Multiple interfaces can be implemented (generally, inherited) at a time.
- Methods inside interfaces are by default public, void and abstract.
- Variables inside interfaces are by default public, static, and final.
- An interface cannot extend a class but it can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.
- The naming convention of a interface usually ends in –able or starts with "Can…" That is because when a class implements an interface, a defined behavior is required to be implemented in implementing classes. So we could say a "Cat" class is "Petable," since Cat has pet methods. But in some cases, if a class name is better without these conventions, and if someone read your code, you think they would understand it better, then go with it!

ENUMS

ENUMS

At the end of the discussion on enums, student will be able to:

- Define and create enums.
- Determine the difference between enums and class.
- Define the methods of enums.

enums

- short for "enumerations", which means "specifically listed“.
- a special "class" that represents a group of constants.
- can have attributes, methods, constructors and instance variables.

enums

- All enums implicitly extend **java.lang.Enum class**. As a class can only extend **one** parent in Java, so an enum cannot extend anything else.

Creating Enum

- use the **enum** keyword
- separate the **constants** with a **comma**
- Note that they should be in **UPPERCASE LETTERS**

EXAMPLE 1:

Weekdays.java

```java
enum Weekdays {
//enumeration constants
   SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
    FRIDAY, SATURDAY;
public static void main(String[] args) {
    Weekdays wd = Weekdays.WEDNESDAY;
        System.out.println("Today is " +wd);
                //wd is the enumeration variable
               //wd can be assigned only the constants defined under enum type Weekdays
     }
}
```

Problems Javadoc Declaration Console

```
<terminated> Weekdays [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe  (Jun 14, 2021, 8:00:46 AM – 8:00:47 AM)
Today is WEDNESDAY
```

EXAMPLE 2:

Main.java

```java
//enum outside the class
enum Weekdays {
SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
    FRIDAY, SATURDAY
}
public class Main {
     public static void main(String[] args) {
    Weekdays wd = Weekdays.WEDNESDAY;
        System.out.println("Today is " +wd);
     }
}
```

Problems Javadoc Declaration Console

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-16.0.1\b
Today is WEDNESDAY
```

EXAMPLE 3:

Main.java

```java
// enum inside a Class
public class Main {
    enum Weekdays {
        SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
        THURSDAY, FRIDAY, SATURDAY
    }
    public static void main(String[] args) {
        Weekdays wd = Weekdays.THURSDAY;
        System.out.println("Tomorrow is " +wd);
    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-16.0.1\

```
Tomorrow is THURSDAY
```

Enum in a Switch Statement

- often used in switch statements to check for corresponding values

Example of Enum in a Switch Statement

Main.java

```java
//enum in switch statement
enum Level {
LOW, MEDIUM, HIGH
}
public class Main {
  public static void main(String[] args) {
    Level lv = Level.MEDIUM;

    switch(lv) {
      case LOW:
        System.out.println("Low level");
        break;
      case MEDIUM:
         System.out.println("Medium level");
        break;
     case HIGH:
        System.out.println("High level");
       break;
    }
  }
}
```

Problems Javadoc Declaration Console

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe
Medium level
```

Differences between Enums and Classes

- Enum constants are **public**, **static** and **final** by default (unchangeable).
- Cannot be used to create objects
- Cannot extend other classes
  (but it can implement interfaces)
- Even though enumeration defines a class type and have constructors, you do not instantiate an **enum** using **new**.

Enum Methods in Java

- values()
- valueof()
- ordinal()
- equals()

values() and valueof() methods

- both are static methods of enum type and can be used to access enum constants.
- **values() method** can be used to return all values present inside enum.
- **valueOf() method** returns the enum constant of the specified string value

Example of values() and valueof()

Main.java

```java
enum Weekdays {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
      class Main {
      public static void main(String args[]) {
    Weekdays wd;
            System.out.println("All constants of enum type Weekdays are:");
                    Weekdays  wdArray[] = Weekdays.values(); //values() method

       for(Weekdays  d : wdArray) //using for loop
                            System.out.println(d);
                    wd = Weekdays.valueOf("WEDNESDAY"); //valueof() method
                            System.out.println("Today is " + wd);
    }
}
```

Problems Javadoc Declaration Console

<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (Jun 14, 2021, 7:54:10 AM – 7:54:11 AM)

```
All constants of enum type Weekdays are:
SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY
Today is WEDNESDAY
```

ordinal() method

- Returns the **order** of an enum instance.
- It represents the sequence in the enum declaration, where the initial constant is aligned an ordinal of ‘ **0** ‘.
- It is like array indexes.

EXAMPLE OF ORDINAL() METHOD

Main.java

```java
enum Color {
 BLACK, PINK, YELLOW, BLUE
}

public class Main {
     public static void main(String []args) {
    Color c = Color.PINK;
         System.out.println(c.ordinal());
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe  (Jun 14, 2021, 7:52:45 AM – 7:52:46 AM)
1
```

equals() method

- It returns true if the specified object is equal to this enum constant.

EXAMPLE OF equals() METHOD

Main.java

```java
enum Color {
    BLACK, PINK, YELLOW, BLUE
}
public class Main {
public static void main(String[] args) {
    Color c1, c2, c3, c4;
                c1 = Color.BLACK;
                            c2 = Color.BLACK;
                            c3 = Color.PINK;
                            c4 = Color.YELLOW;

if(c1.equals(c2))

    System.out.println("Colors are the same.");
 else
    System.out.println("Colors are different.");
     }
}
```

Problems @ Javadoc Declaration Console

<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (Jun 14, 2021, 7:51:39 AM – 7:51:39 AM)

```
Colors are the same.
```

Enumeration with Constructor, instance variable, Method And Enum values

- To hold the value of each constant you need to have an **instance variable** (generally, **private**).
- You cannot create an object of an enum explicitly so, you need to **add a parameterized constructor to initialize the value(s).**
- The **initialization** should be **done only once.** Therefore, the **constructor** must be **declared private or default**.
- To returns the values of the constants using an instance **method(getter).**

Example of Enumeration with Constructor, instance variable, Method And Enum values

Main.java

```java
enum Transport {
   //Constants with values
   PLANE(600), TRAIN(100), AUTOMOBILE(80);
   //Instance variable
   private int speed;
   //Constructor to initialize the instance variable
   Transport(int speed) {
      this.speed = speed;
   }
    //getter method
   public int getSpeed() {
      return this.speed;
   }
}
public class Main{
   public static void main(String args[]) {
      Transport tp = Transport.PLANE;
         System.out.println("Speed of "+Transport.PLANE+" is: "+tp.getSpeed());
      }
   }
```

Problems | Javadoc | Declaration | Console

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe  (Jun 14, 2021, 7:49:57 AM – 7:49:58 AM)
Speed of PLANE is: 600
```

Implement an Interface using an Enum

Main.java

```java
//Implement an Interface using an Enum
// Defining an interface
interface week {

    public int day();
}
// Initializing an enum which implements the above interface
enum Day implements week {
  // Initializing the possible days
SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;

    public int day() {
        return ordinal();
    }
}
// Main Class
public class Main{
    public static void main(String args[]) {
        System.out.println("It is day number " + Day.WEDNESDAY.day() + " of a week.");
   }
}
```

Problems Javadoc Declaration Console

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe  (Jun 14, 2021, 7:47:05 AM – 7:47:08 AM)
It is day number 3 of a week.
```

Enum with abstract method

TestEnums.java

```java
enum TrafficSignal {
//enum with abstract method
    RED{
        @Override
        public void action(){
            System.out.println("STOP");
        }
    },
    GREEN{
        @Override
        public void action(){
            System.out.println("GO");
        }
    },
    ORANGE{
        @Override
        public void action(){
            System.out.println("SLOW DOWN");
        }
    };


    public abstract void action();
}


public class TestEnums{

    public static void main(String []args){
       TrafficSignal signal = TrafficSignal.RED;
       signal.action();

    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> TestEnums [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (Jun 14, 2021, 7:43:27 AM – 7:43:28 AM)

```
STOP
```

When To Use Enums?

- Use **enums** when you have values that you know **aren't going to change**, like months, days, colors, deck of cards, etc.

USER INPUT

USER INPUT

- The **Scanner** class is used to get user input, and it is found in the **java.util** package.

- To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation.

INPUT TYPES

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

EXAMPLES

```
import java.util.Scanner;

class Main {
        public static void main(String[] args) {
                Scanner myObj = new Scanner(System.in);

                        System.out.println("Enter name, age and salary:");

                String name = myObj.nextLine();
                int age = myObj.nextInt();
                double salary = myObj.nextDouble();

                        System.out.println("Name: " + name);
                        System.out.println("Age: " + age);
                        System.out.println("Salary: " + salary);
  }
}
```

Explanation

- Based on the last slide the output will be user dependent:

Result:

```
Enter name, age and salary:
Darkin
19
1000
```

Result:

```
Enter name, age and salary:
Darkin
19
1000
Name: Darkin
Age: 19
Salary: 1000
```

OUTPUT

REFERENCES:

Head First Java 2nd Edition (Book)

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

https://dzone.com/articles/interface-default-methods-java

https://www.coursera.org/learn/object-oriented-java/lecture/lc1ml/core-abstract-classes-and-interfaces

https://docs.oracle.com/javase/8/docs/api/

Stackoverflow

https://www.w3schools.com/java/java_enums.asp

https://www.studytonight.com/java/enumerations.php

https://www.tutorialspoint.com/Enum-in-Java

https://www.google.com/amp/s/www.geeksforgeeks.org/enum-in-java/amp/

Java User Input (Scanner class) (w3schools.com)

https://docs.oracle.com/javase/8/docs/api/

ACTIVITY: ROCK, PAPER, SCISSORS!

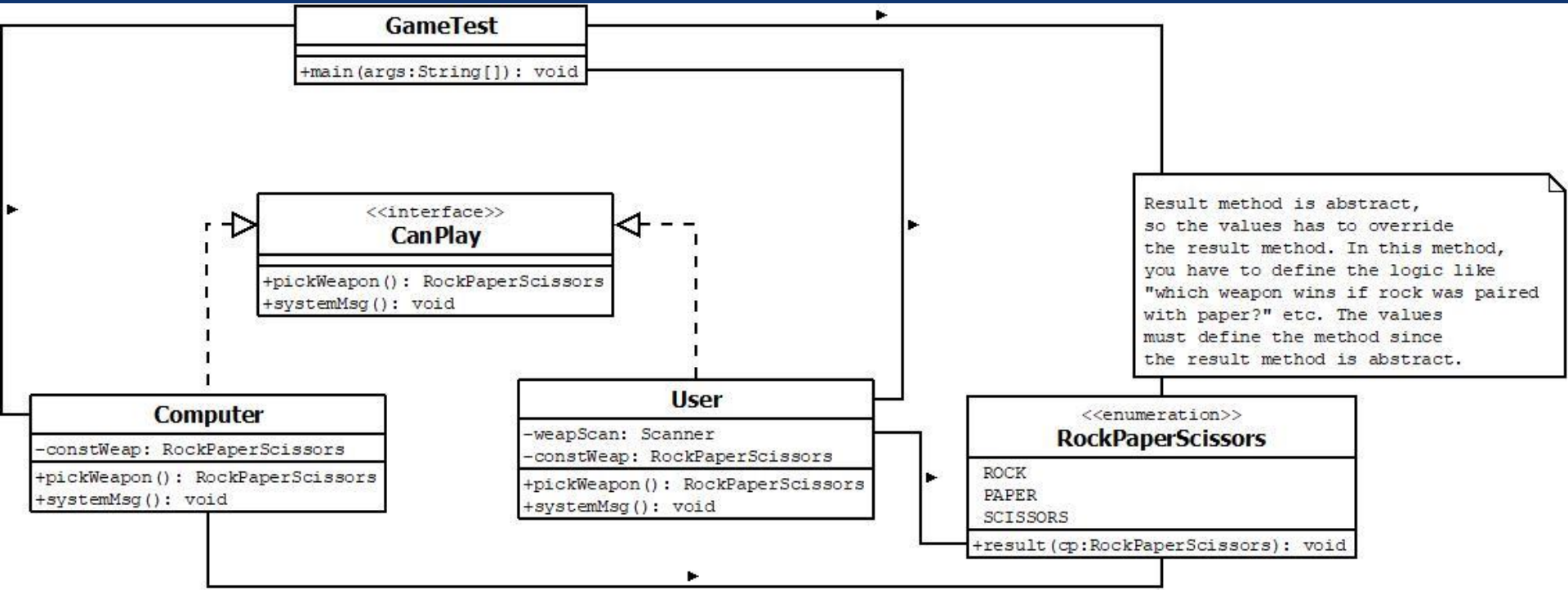
- Create a enum named “RockPaperScissors”. It has values ROCK, PAPER, and SCISSORS. Inside that enum, create an abstract void method called result with parameter (RockPaperScissors *any variable name*). The values stated above must override this method. Inside the method, write the logic of “who wins if computer picked a ROCK, PAPER, OR SCISSOR.” The method has public access modifier.

- Create an interface called “CanPlay”. Inside, create a pickWeapon() method of type RockPaperScissors, and another method named systemMsg() with void return type.

- Create a class named Computer which implements CanPlay interface. In the field, declare a variable constWeap of RockPaperScissors reference with private access. Override the methods, then create a random logic behavior inside it. As an example, if (1), then constWeap = *enumname*.valueOf(*ROCK*), then continue. Inside the systemMsg method, print a message that display if its the computer’s turn to pick weapon.

- Create a class named User which implements CanPlay interface. Do the same for Computer method, but instead, it accepts user input. Declare the scanner and the constWeap variable(similar to computer) in the field. Make all field variables private. Pass the input in the valueOf(*userinput*) parameter.

- In the main class, create a Computer and User object. Call the method and store the values they return in RockPaperScissors reference. Then call the method result in RockPaperScissors enum.
  E.g. *uservarname*.result(*computervarname*); or *usermethod*.result(*computermethod*)


GameTest
+main(args:String[]): void
<<interface>>
CanPlay
+pickWeapon(): RockPaperScissors
+systemMsg(): void
Result method is abstract, so the values has to override the result method. In this method, you have to define the logic like "which weapon wins if rock was paired with paper?" etc. The values must define the method since the result method is abstract.
Computer
-constWeap: RockPaperScissors
+pickWeapon(): RockPaperScissors
+systemMsg(): void
User
-weapScan: Scanner
-constWeap: RockPaperScissors
+pickWeapon(): RockPaperScissors
+systemMsg(): void
<<enumeration>>
RockPaperScissors
ROCK
PAPER
SCISSORS
+result(cp:RockPaperScissors): void

Problems | Javadoc | Declaration | Console

```
<terminated> GameTest (1) [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe (Jun 16, 2021, 10:21:11 AM)
Your turn to choose....
Pick a Weapon
ROCK
Computer's turn to choose....
Computer picked: SCISSORS
You Win
```