Part 2 Java

Lecture 51:

Super Method

- By default, every constructor has super method as its first statement.
- super ();
- Means it call the constructor of the super class.

```
class B extends A
{
    public B()
    {
        super();
        System.out.println(x: "in B");
    }
    public B(int n)
    {
        super();
        System.out.println(x: "in B int");
    }
}
```

Note: Every class in java extends from Object class.

```
class A extends Object
{
    public A()
    {
        super();
        System.out.println(x: "in A");
    }
```

This method

- It will execute the constructor of same class.

```java
class B extends A
{
    public B()
    {
        super();
        System.out.println(x: "in B");
    }
    public B(int n)
    {
        this();
        System.out.println(x: "in B int");
    }
}
```

Lecture 52:

Method overriding

- Method of child class override the method of parent class with same name.

```java
Hello.java > ...
1   class calc{
2       public int add(int n1 , int n2){
3           return n1+n2;
4       }
5   }
6   class Advcalc extends calc{
7       public int add(int n1 , int n2){
8           return n1+n2+1;
9       }
10  }
11  class Hello {
        Run main | Debug main
12      public static void main(String [] arg){
13          Advcalc obj = new Advcalc();
14          int result = obj.add(4, 5);
15          System.out.println(result);
16      }
17  }
18
```

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

9
PS D:\Java\Telusko> javac Hello.java
PS D:\Java\Telusko> java Hello
10
PS D:\Java\Telusko>
```

Lecture 53:

Packages in Java

- Packages are folders.
- Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.
- To a file add into package :- package Package_name;
- Import files of packages :- import package_name.class_name;
- Import all file of apackage:- import package_name.*;
- ArrayList is from a package java.util.ArrayList

Note: whatever Class you use in java belongs to a package.

- Every java file have java.lang package imported in it which contain classes like : System

Lecture 54:

Access Modifier

Access modifiers help to restrict the scope of a class, constructor, variable, method, or data member.

It provides security, accessibility, etc to the user depending upon the access modifier used with the element.

Public: can be accessed from anywhere.

Private: can't not be accessed outside the class.

Default: can be accessed in same package.

Protected: Can't be accessed outside the package.

- If it is a sub class the can be accessed.

Which one to use ?

- Try to make your classes public
- Try to keep your instance variable private.
- Methods most of the time will be public .
- Want to accesses only in sub class outside the package make it Protected
- Avoid default.

Lecture 55:

Polymorphism

- Many behaviors
- Polymorphism allows us to perform a single action in different ways.
- Poly – Many and Morph – Forms

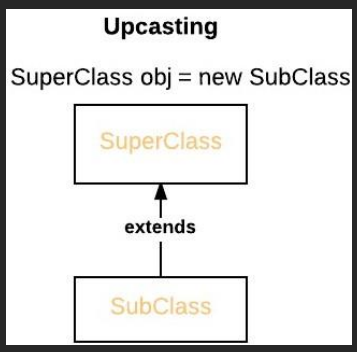
Types of Java Polymorphism

1. Compile time: which method will be executed decide at compile time
    - Achieved by function Overloading
2. Run time: which method will be executed decide at run time
    - Achieved by function Overriding.
    - It is also known as Dynamic Method Dispatch.

Lecture 56:

Dynamic Method Dispatch

- Implementation of run time polymorphism.
- A superclass reference variable can refer to a subclass object. This is also known as upcasting.



- Dynamic method dispatch: It is a process through which a call to overridden method is resolved during runtime rather than compile time.

```java
class A{
    public void show(){
        System.out.println(x:"in show A");
    }
}
class B extends A{
    public void show(){
        System.out.println(x:"in show B");
    }
}
class C{
    public void show(){
        System.out.println(x:"in show C");
    }
}
class Hello {
    Run main | Debug main | Run | Debug
    public static void main(String [] arg){
        A obj = new A();
        obj.show();

        obj = new B();
        obj.show();

        // obj = new C(); // error because C class is not
        child of A. We can assign Object of child class to
        refernce variable of parent class
    }
}
```

Lecture 57:

Final Keyword

- Can be used with : variable, method and class

Final Variable: final make variable constant.

- Ex :

```java
public class Demo
{
    public static void main(String a[])
    {
        final int num = 8;
        num = 9;
        System.out.println(num);


    }

}
```

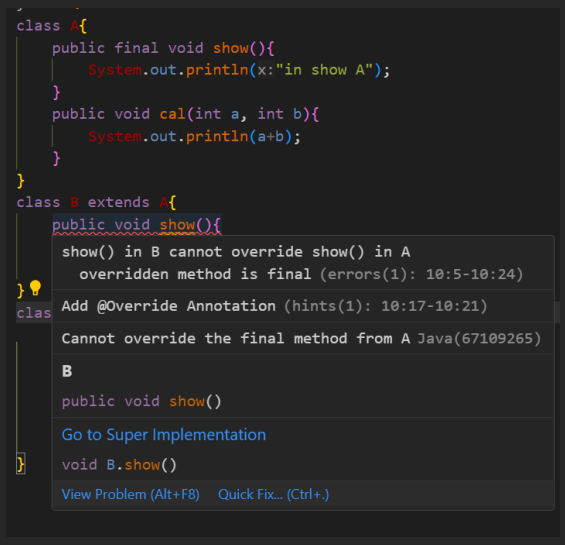
Final Class:

- Final class can’t be inherited.

Final Method:

- Final method can’t be Overridden.

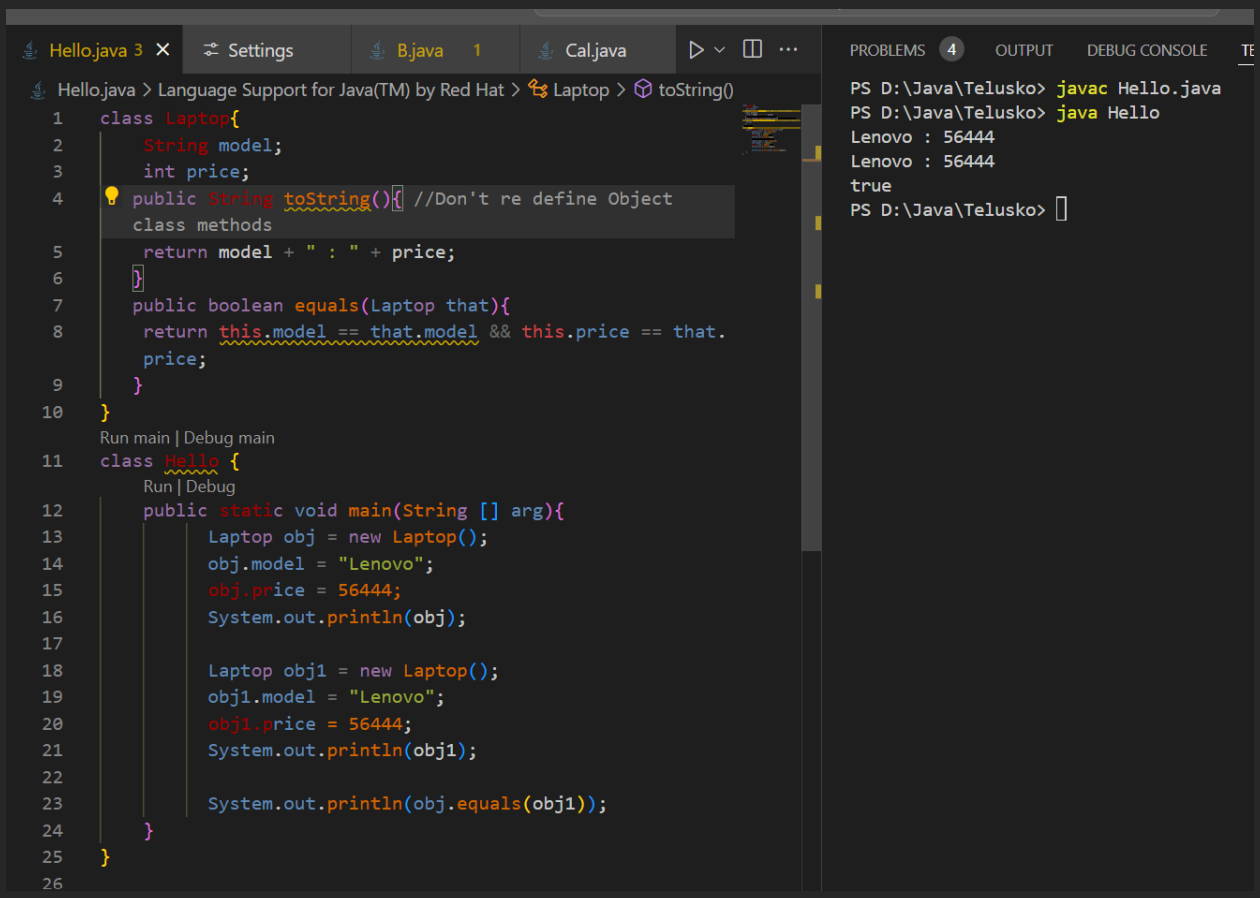
Ex:



Lecture 58:

Object Class

- Every class of java extends the object class.
- Every time you print an object it will call to string method even you don’t mention. System.out.println(obj.toString( ));
- And he explain how to work with Object class method

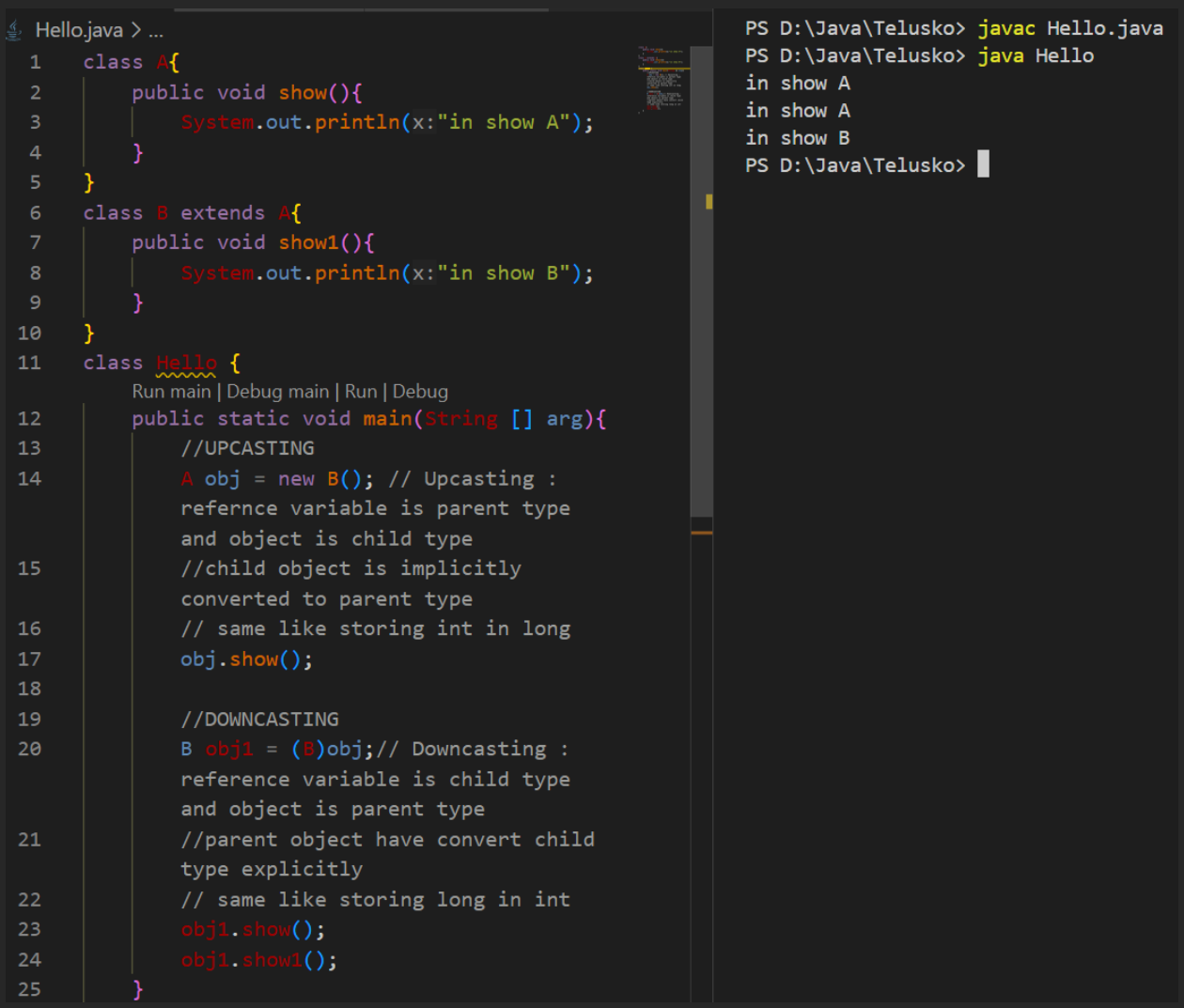


- Watch that video again if want understand more in 1.5x

Lecture 59:

Upcasting and Downcasting

Typecasting, also known as type conversion, is the process of changing the value of one data type into another.



Lecture 60:

Java is Not Purely Object-Oriented Language

Why -

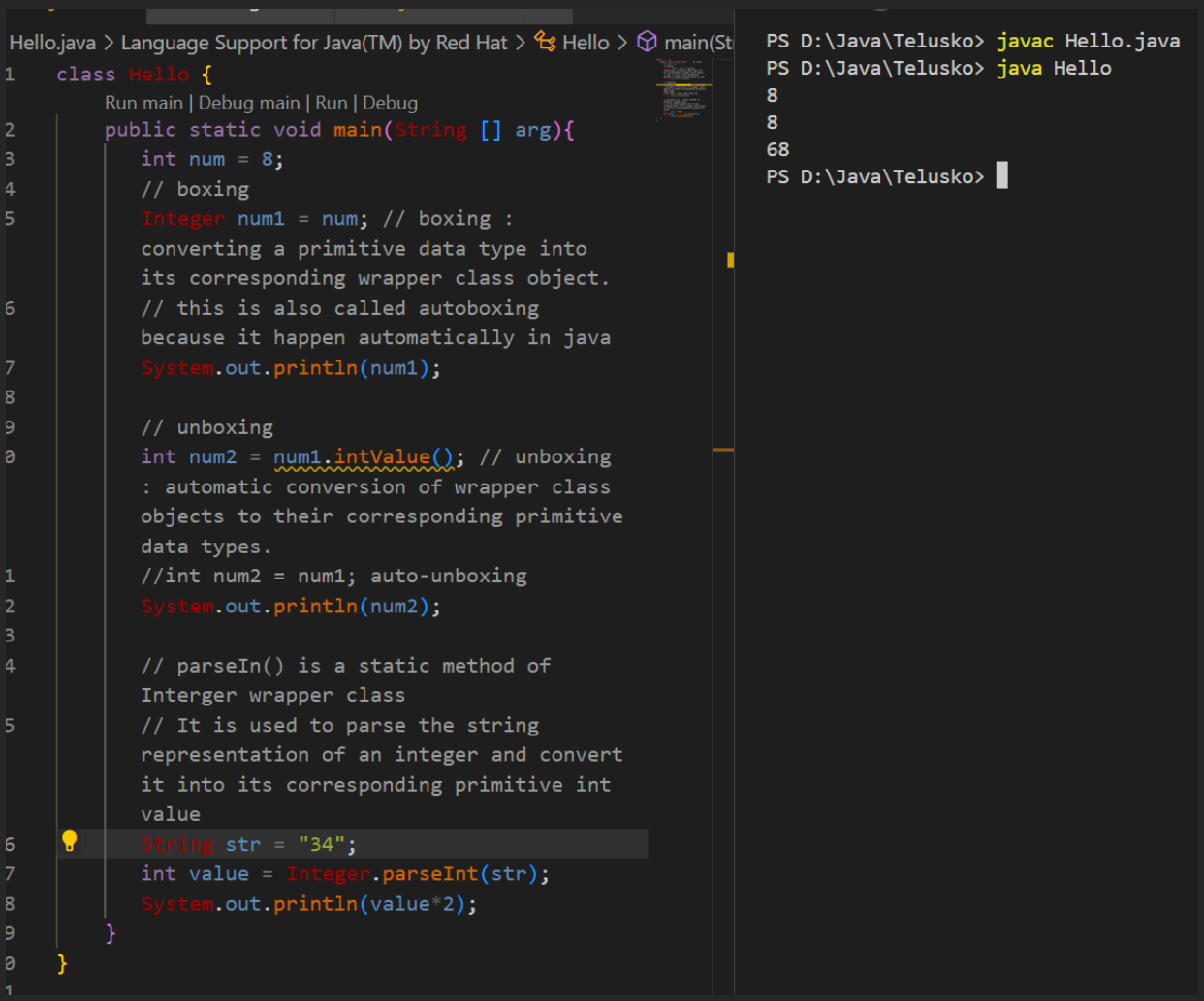
1. Java Supports primitive data types like – byte, short, int, long, float, double, char, and Boolean. Which are not extending from any class. Means are not object.
   - It helps java to improve ProFormance

2. Java supports static methods and variables:
   - Static methods and variables can be accessed without creating an object of the class. This violates the principle of encapsulation, which is one of the fundamental principles of object-oriented programming.

Wrapper Classes

- In Java, there are eight primitive data types: byte, short, int, long, float, double, char, and boolean.
- Wrapper classes provide a way to treat primitive data types as objects.
- This allows you to use the primitive data types in places where only objects are accepted, such as in collections frameworks.
- wrapper classes are Integer, Double, Boolean, Character, Byte, Short, Long, and Float.

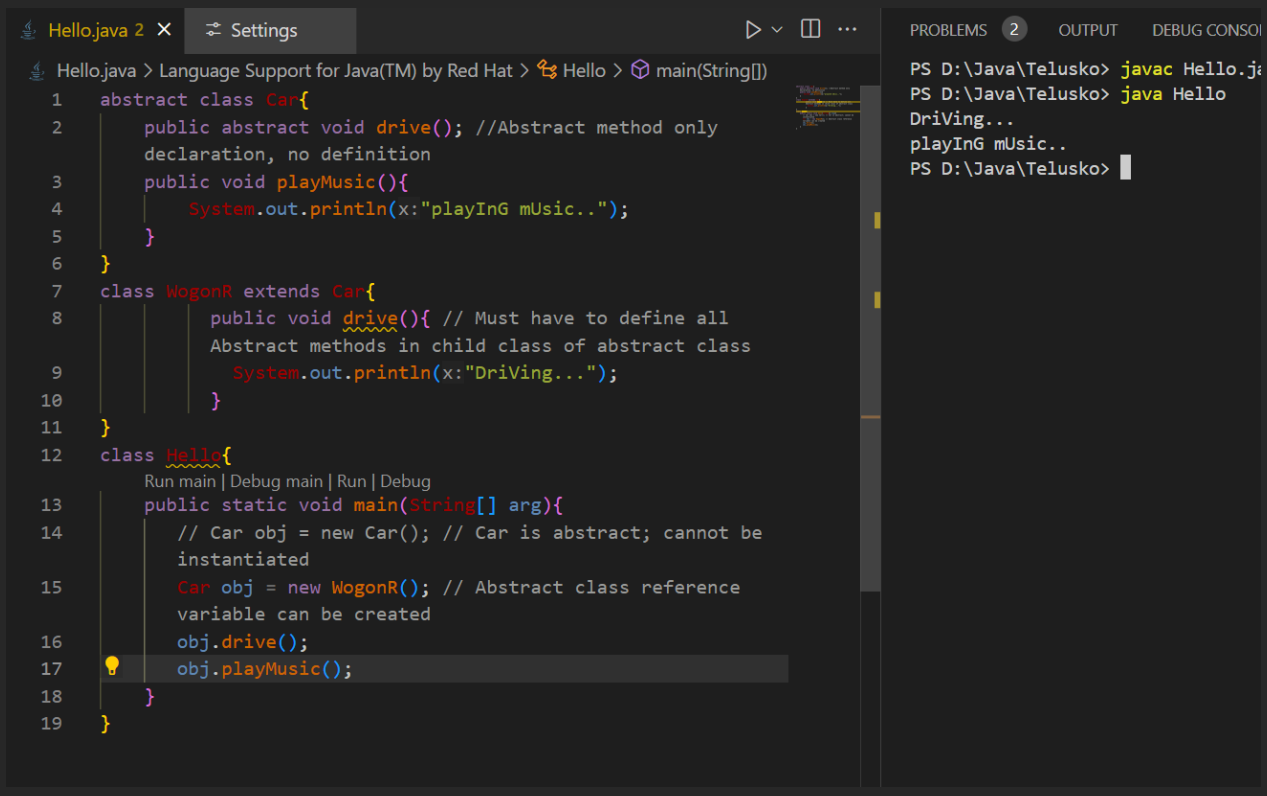
Boxing, unboxing and parseInt( );



Lecture 61:

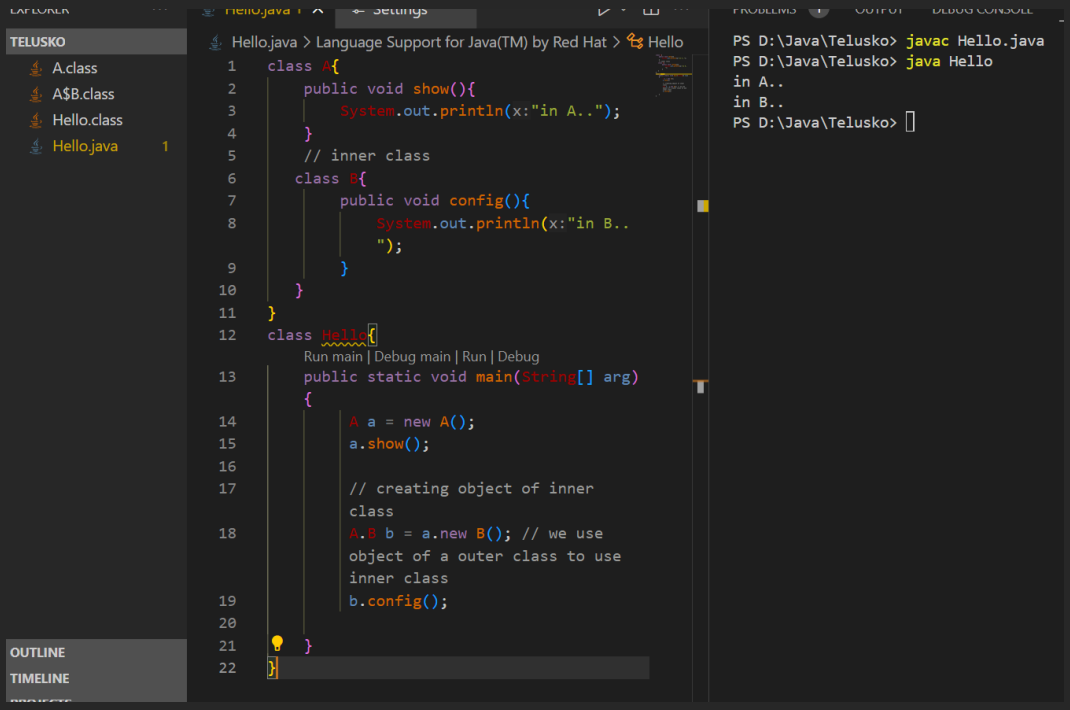
Abstract Keyword

- An `abstract` method belongs to an `abstract` class, and it does not have a body. The body is provided by the subclass.
- Subclass which has the abstract method definition know as- concrete class
- We can't create object of abstract class.



Lecture 61:

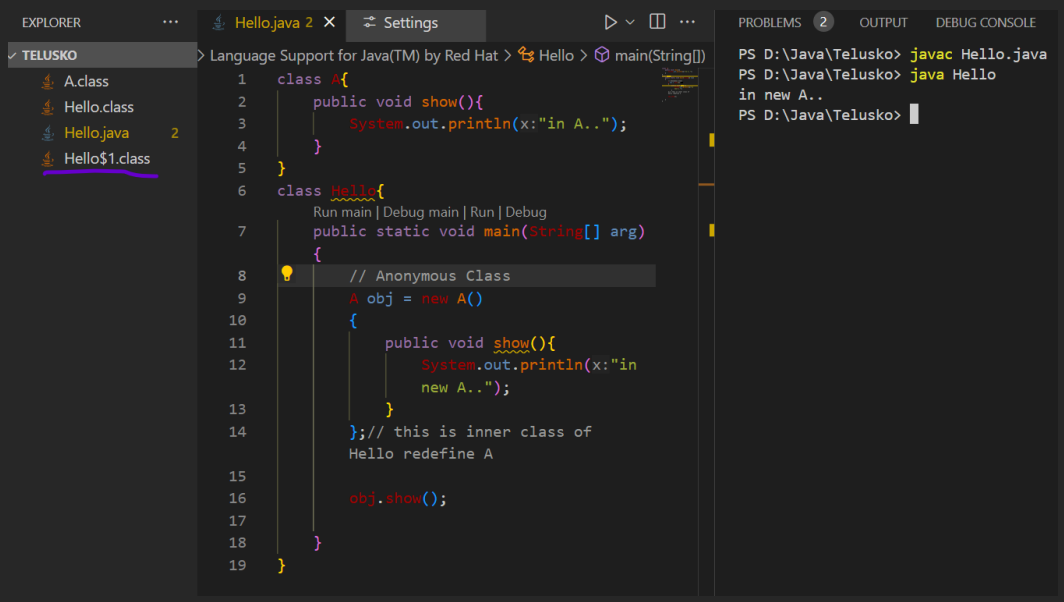
Inner Class



- Use static for inner class, so no need create object of outer class to use inner class.
- Outer class can't be static.

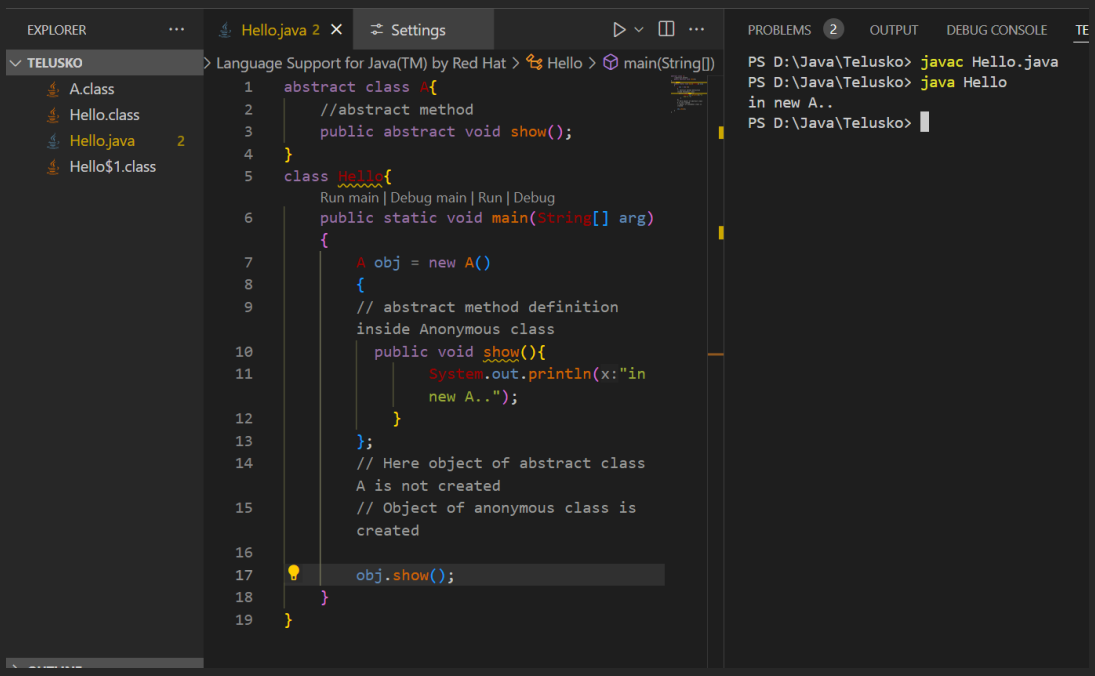
Lecture 62:

Anonymous Class



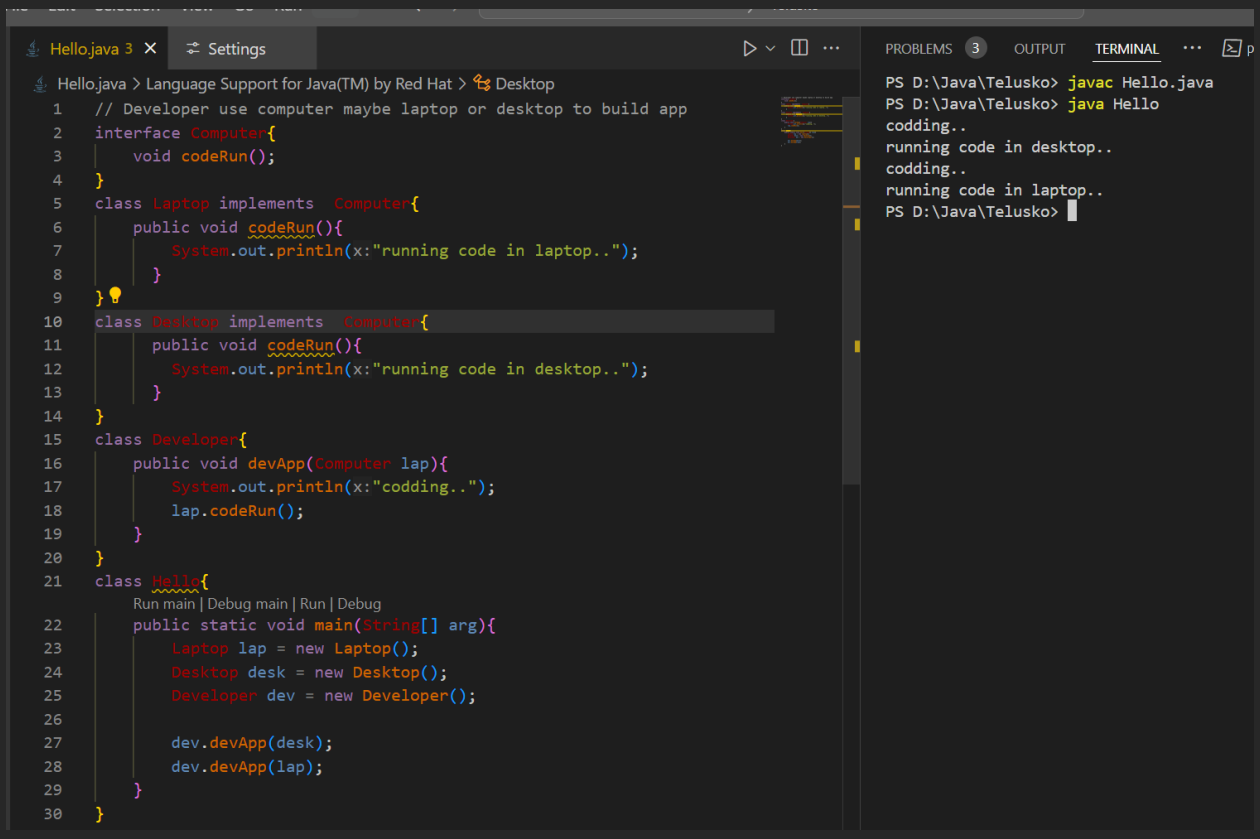
Lecture 63:

Abstract and Anonymous inner class



Lecture 65:

Need of Interface in Java



Lecture 66:

What is Interface?

Interfaces

Another way to achieve abstraction in Java, is with interfaces.

An `interface` is a completely "**abstract class**" that is used to group related methods with empty bodies:

Example

Get you

```
// interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
```

- Interface is not a class
- Abstraction in Java is the process in which we only show essential details/functionality to the user. The non-essential implementation details are not displayed to the user.
- To access the interface methods, the interface must be "implemented" (kind of like inherited) by another class with the `implements` keyword (instead of `extends`)

Notes on Interfaces:

- Like **abstract classes**, interfaces **cannot** be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default `abstract` and `public`
- Interface attributes are by default `public`, `static` and `final`
- An interface cannot contain a constructor (as it cannot be used to create objects)

Why And When To Use Interfaces?

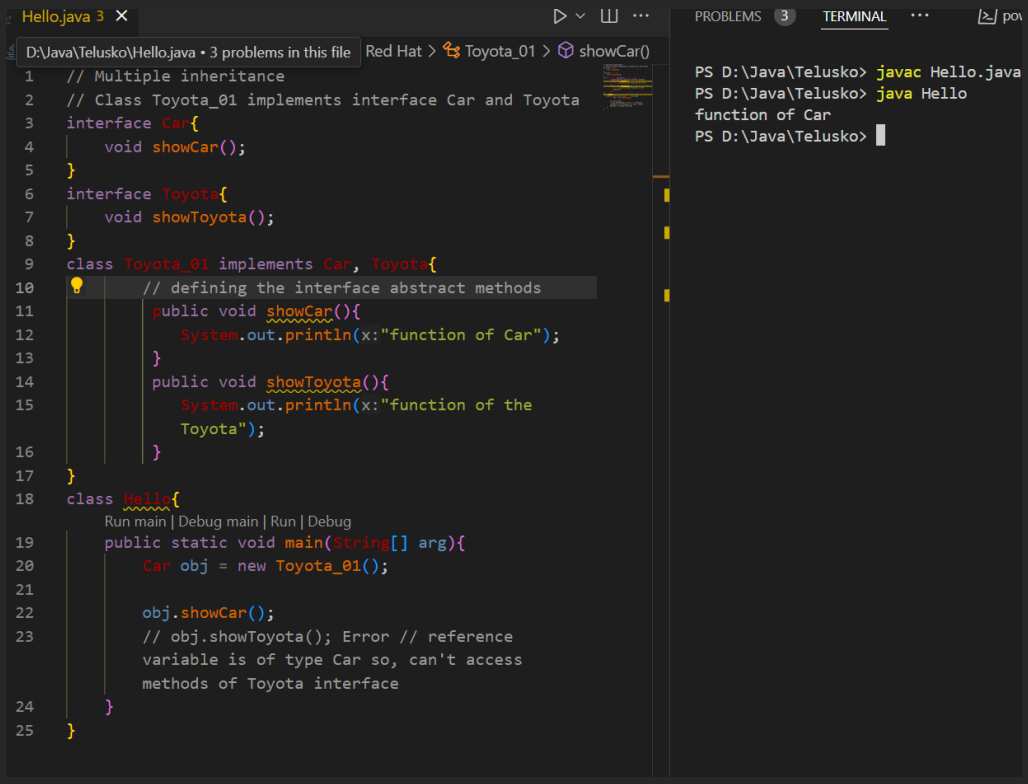
1) To achieve security - hide certain details and only show the important details of an object (interface).

2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces. **Note:** To implement multiple interfaces, separate them with a comma (see example below).

- attribute is a variable that is declared within a class. Also known as instance variable.

Lecture 67:

More on Interface



Lecture 68:

Enum (Enumeration)

- Used to create named constants.

Enums

An `enum` is a special "class" that represents a group of **constants** (unchangeable variables, like `final` variables).

To create an `enum`, use the `enum` keyword (instead of class or interface), and separate the constants with a comma. Note that they should be in uppercase letters:

Example

Get your own Java Server

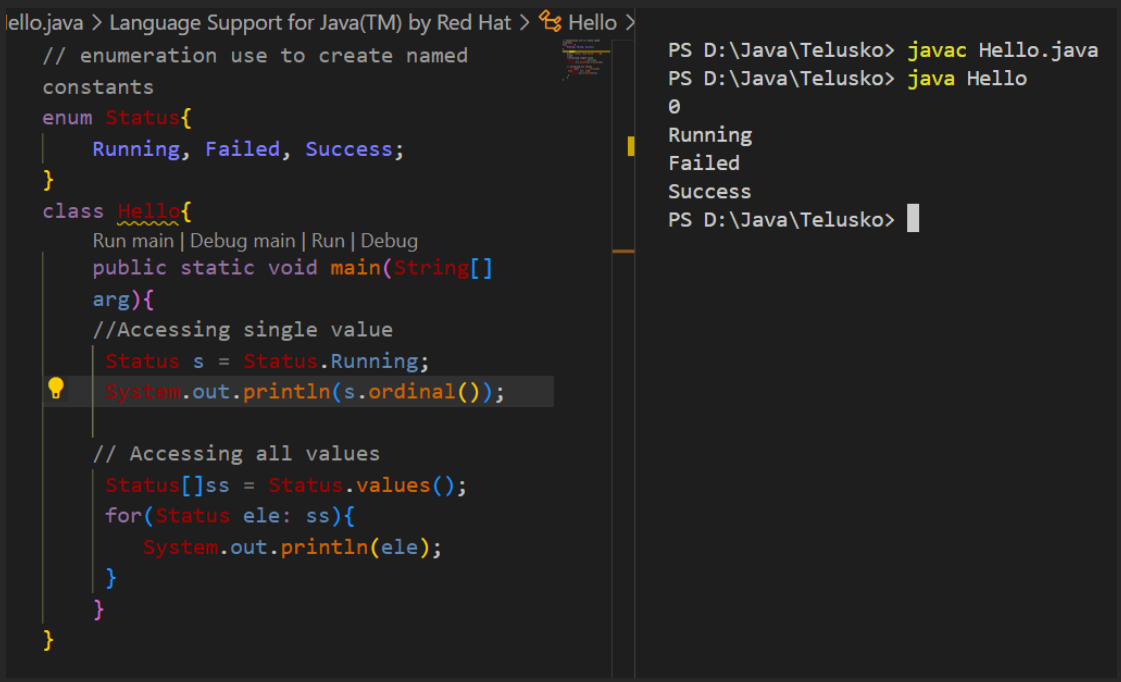
```
enum Level {
  LOW,
  MEDIUM,
  HIGH
}
```

You can access `enum` constants with the **dot** syntax:

```
Level myVar = Level.MEDIUM;
```

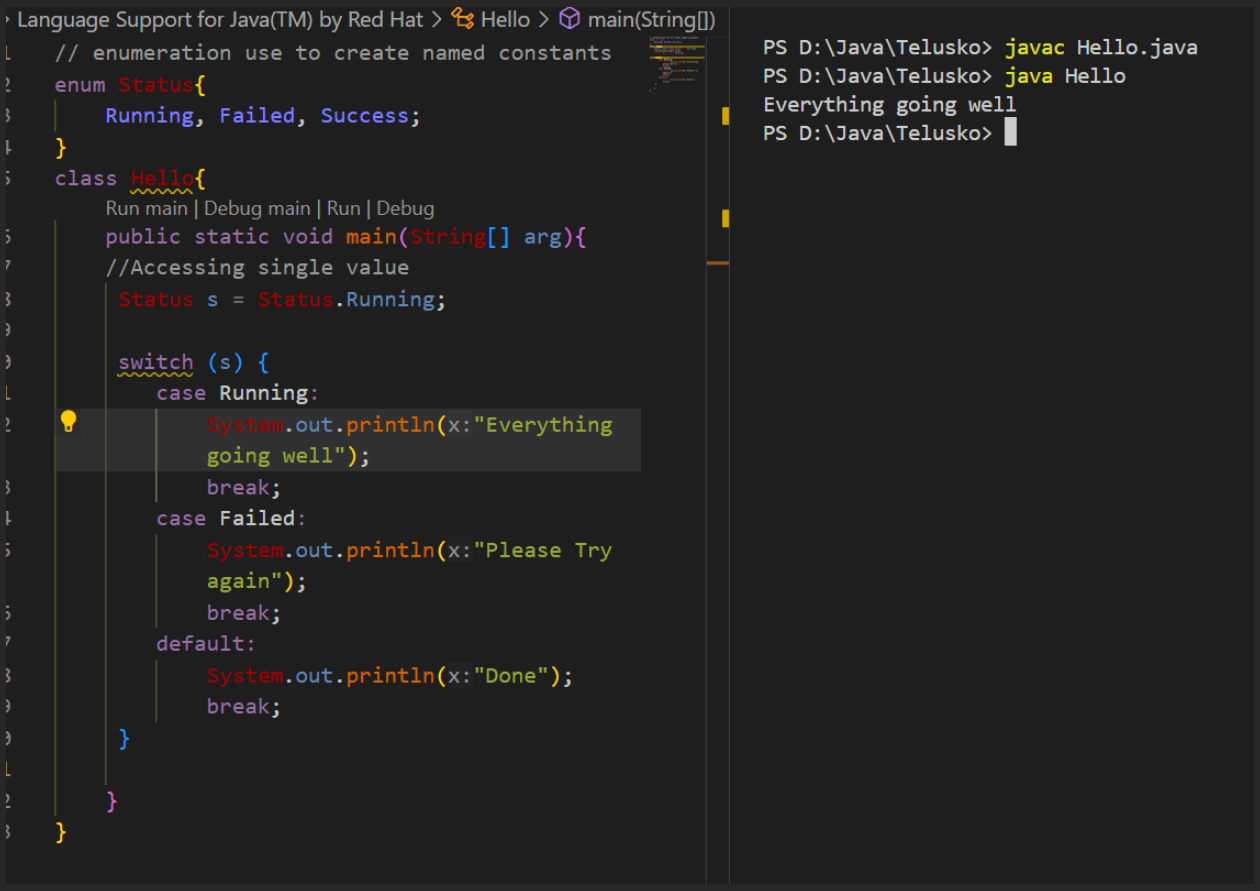
Try it Yourself »

Enum is short for "enumerations", which means "specifically listed".



Lecture 69:

Enum in Switch



Lecture 70:

Enum Class In java

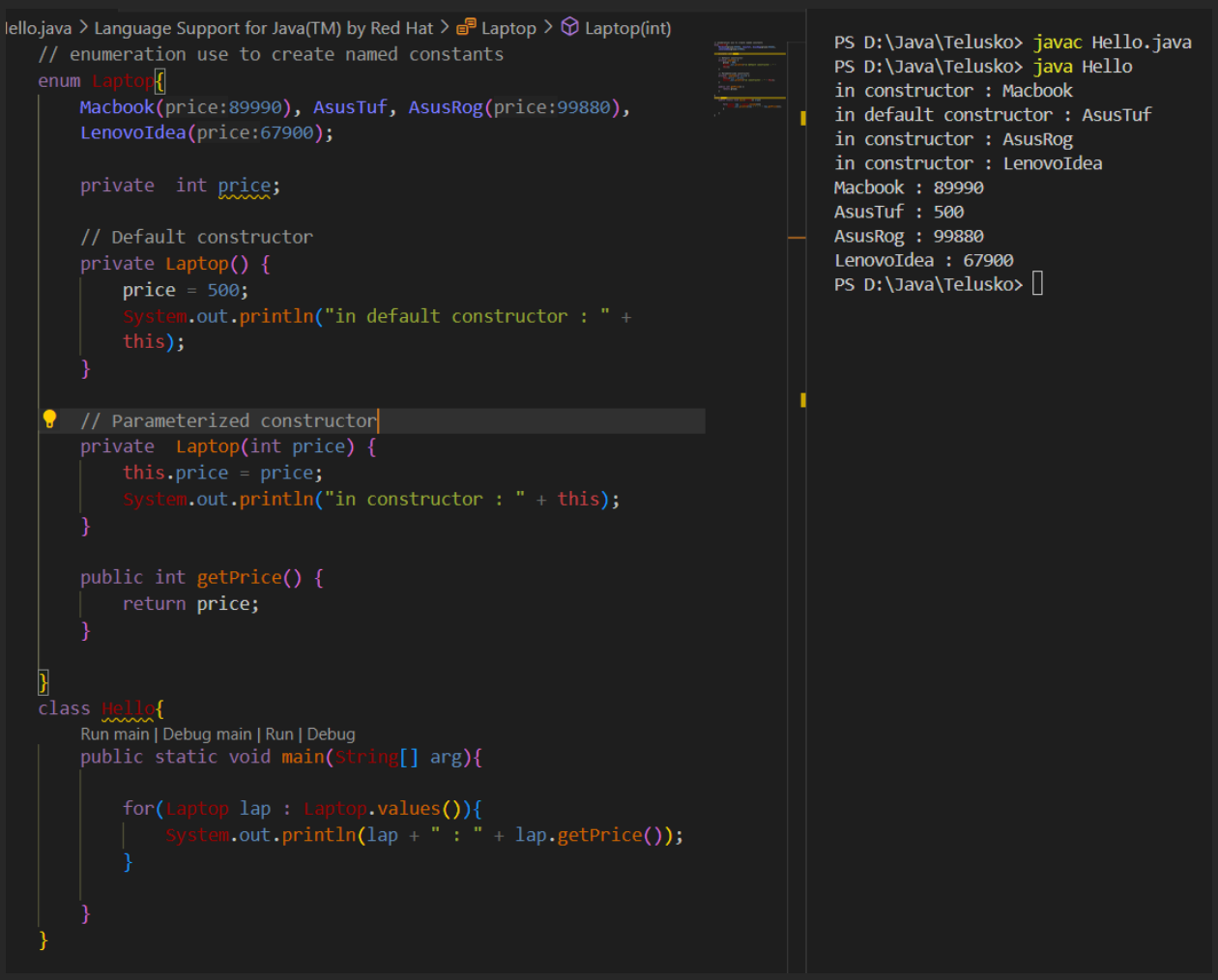
Difference between Enums and Classes

An `enum` can, just like a `class`, have attributes and methods. The only difference is that enum constants are `public`, `static` and `final` (unchangeable - cannot be overridden).

An `enum` cannot be used to create objects, and it cannot extend other classes (but it can implement interfaces).

Why And When To Use Enums?

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


lello.java > Language Support for Java(TM) by Red Hat > Laptop > Laptop(int)
// enumeration use to create named constants
enum Laptop{
Macbook(price:89990), AsusTuf, AsusRog(price:99880),
LenovoIdea(price:67900);
private int price;
// Default constructor
private Laptop() {
price = 500;
System.out.println("in default constructor : " +
this);
}
// Parameterized constructor
private Laptop(int price) {
this.price = price;
System.out.println("in constructor : " + this);
}
public int getPrice() {
return price;
}
}
class Hello{
Run main | Debug main | Run | Debug
public static void main(String[] arg){
for(Laptop lap : Laptop.values()){
System.out.println(lap + " : " + lap.getPrice());
}
}
}
PS D:\Java\Telusko> javac Hello.java
PS D:\Java\Telusko> java Hello
in constructor : Macbook
in default constructor : AsusTuf
in constructor : AsusRog
in constructor : LenovoIdea
Macbook : 89990
AsusTuf : 500
AsusRog : 99880
LenovoIdea : 67900
PS D:\Java\Telusko>