Crio Sprint: JAVA-112

Session 4 - OOPs: Inheritance

Session Agenda

- Static keyword
- Packages
- Math Library
- Inheritance

Static keyword

How would we do this in Java?

- We want to **have a common variable across all instances of a class**, like the Company Name for an employee class.
  - This can be used to **keep count of number of instances created for a class**.
- We want to create a **method that is not related to a particular class instance but provides stand alone functionality**.
  - Example: Find the greater of two passed values.

Ans: **static keyword**, let's get into more details.

Java static keyword

- It’s a member of a class **that isn’t associated with a specific instance** of the class.
- Can be **accessed without creating a new class instance**.
- A static member is **shared among all the instances** of the class.
- Two important static members are:
  - **static variable/field**
  - **static method**

Static variable

- It's value is **common for all instances** of the class.
- It **gets memory only once** in the class area.
- Check Math.PI in the Math Java API and you'll find:
    - public static final double PI = 3.141592653589793;
    - Marked **public**, so accessible everywhere.
    - Marked **static**, so Math instance creation can be avoided.
    - Marked **final** (Will discuss it further).

What will be the output?

```
class Counter{
    static int count=0;//will get memory only once 
and retain its value

    Counter(){
        count++;//incrementing the value of static 
variable
        System.out.println(count);
    }

    public static void main(String args[]){
        //creating objects
        Counter c1=new Counter();
        Counter c2=new Counter();
        Counter c3=new Counter();
    }
}
```

Java static method

- A static method means "behavior not dependent on an instance variable, so no instance/object is required. Just the class."
- Can be invoked without the need for any instance.
- Can access static member variable and modify it.
- Check Math Class in the Math Java API and you'll find:
    - Math.min(), Math.max() ,etc.

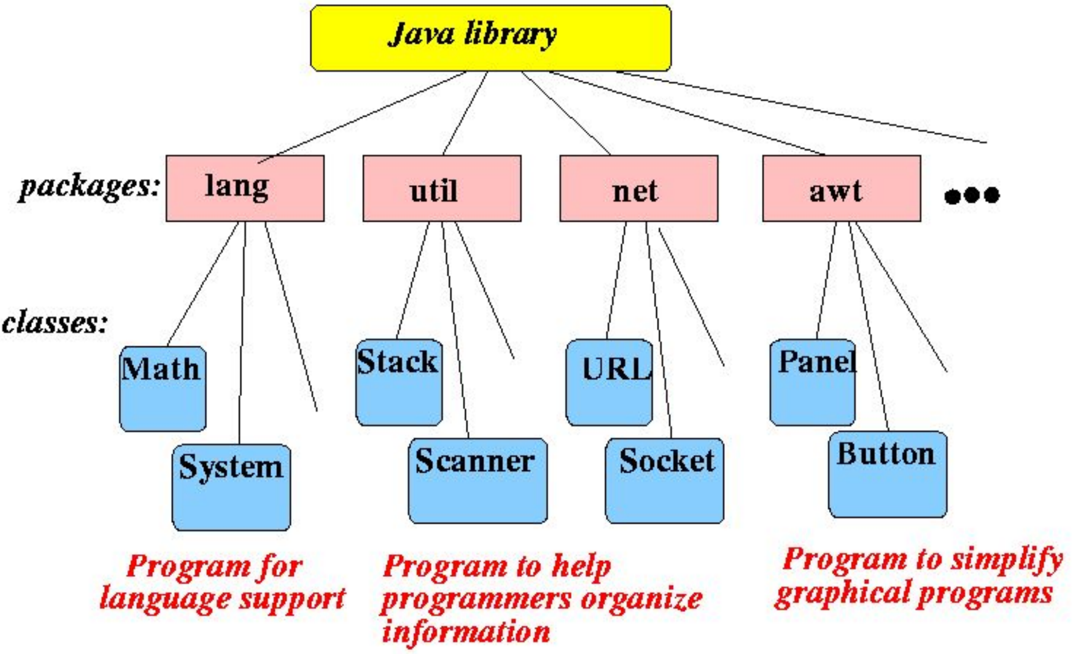
```
class Calculate{
    // static method
    static int cube(int x){
        return x*x*x;
    }

    public static void main(String args[]){
        int result=Calculate.cube(5);
       System.out.println(result);
    }
}
```

Packages

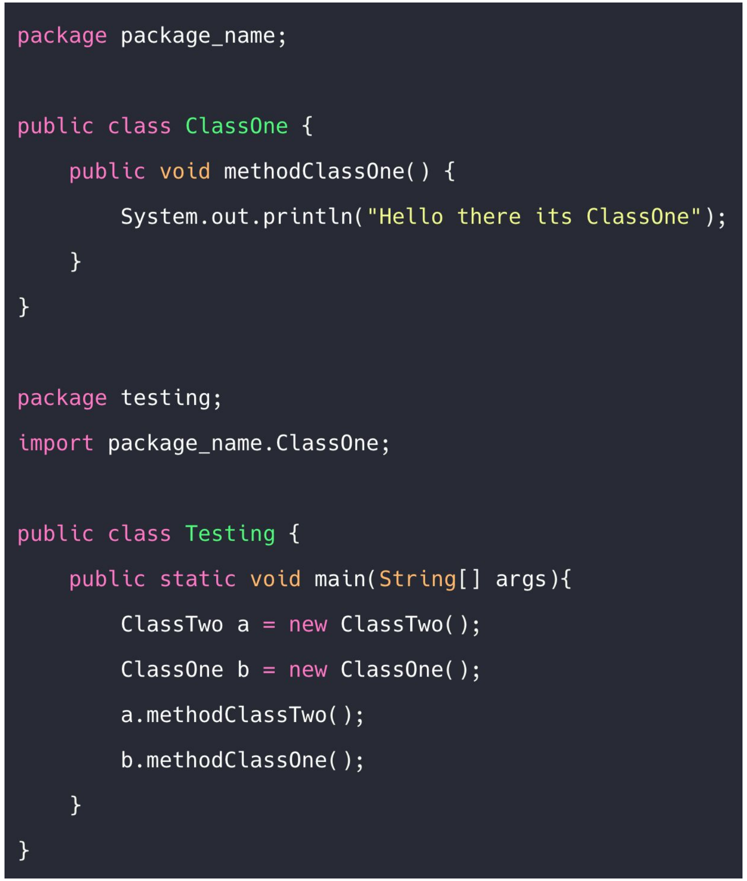
Java libraries

- What is a library / **package** in Java?
  - A package is a way to organize related functionality or code in a single place in Java.
  - Consists of a set of classes that can be imported and used in other pieces of code.
- How to use a package?
  - Use **import** to include the package in your code
  - Then invoke methods on those package classes
- Standard Java Packages/Libraries
  - Java.lang (.math, .System etc.) - Remember the System.out you've been using all along?
  - Java.util (.Random, .Scanner etc.)

Packages


Java library
packages:
lang
util
net
awt
●●●
classes:
Math
System
Stack
Scanner
URL
Socket
Panel
Button
Program for language support
Program to help programmers organize information
Program to simplify graphical programs

In-built Java Package



```
package package_name;

public class ClassOne {
    public void methodClassOne() {
        System.out.println("Hello there its ClassOne");
    }
}

package testing;
import package_name.ClassOne;

public class Testing {
    public static void main(String[] args){
        ClassTwo a = new ClassTwo();
        ClassOne b = new ClassOne();
        a.methodClassTwo();
        b.methodClassOne();
    }
}
```

User Defined Package

Math Library

Java Library - Math class

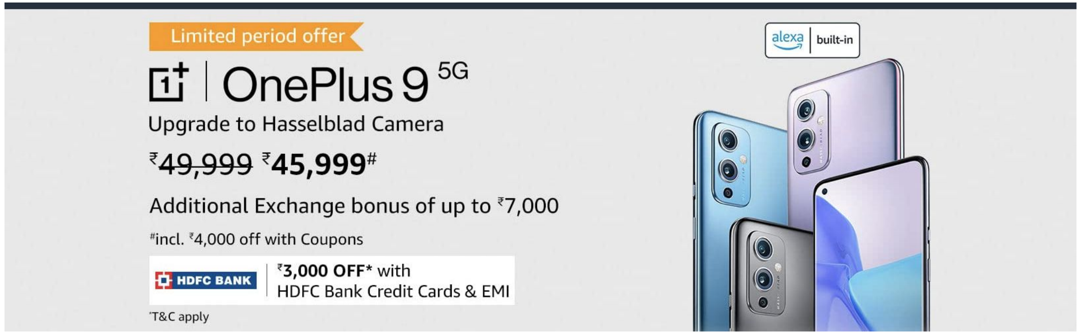
- java.lang.math https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html
    - sqrt(double) - Returns the correctly rounded positive square root of a double value
    - max() - Can take two double, float, int or long values and return the maximum of the two.
    - min() - Can take two double, float, int or long values and return the minimum of the two.
    - random() - Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0
    - round() - Can take a double/float and return the closest long/int
    - pow(double,double) - Returns the value of the first argument raised to the power of the second argument
    - abs() - Can take two double, float, int or long values and return the absolute value of it.
    - ceil(double) - Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer.
    - floor(double) - Same as ceil, but returns largest (closest to +ve infinity) less than or equal to ..
    - ...

Inheritance

Scenario 1

- How many of you regularly shop on Amazon, Flipkart or any other e-commerce website?
- Do you use Credit Card for payment on these sites?

- Ever seen an offer like this?


Limited period offer

OnePlus 9 5G

Upgrade to Hasselblad Camera

~~₹49,999~~ ₹45,999#

Additional Exchange bonus of up to ₹7,000

#incl. ₹4,000 off with Coupons

HDFC BANK | ₹3,000 OFF* with HDFC Bank Credit Cards & EMI

*T&C apply

Scenario 1

- What do you use the Credit Card for?
- What are the basic features provided by any Credit Card?
  - Online/Offline Transactions
  - Emergency Loan
  - Avail Discount Benefits
  - Earn Reward Points
  - ...
- Who can provide you with a credit card?

Union Bank of India
AXIS BANK
State Bank of India
punjab national bank
बैंक ऑफ़ बड़ौदा
Bank of Baroda
IDBI BAN
ICICI Bank
HDFC BANK
Canara Bank
बैंक ऑफ़ इंडिया

Scenario 1

- How are these credit cards different from the basic credit card?
- For payment with these cards, on their respective websites:
  - 5% of the total amount is provided as cashback in Amazon Pay Wallet.
  - 4% is provided as cashback as Flipkart Super Coins.
- Are the above features present in every basic credit card?

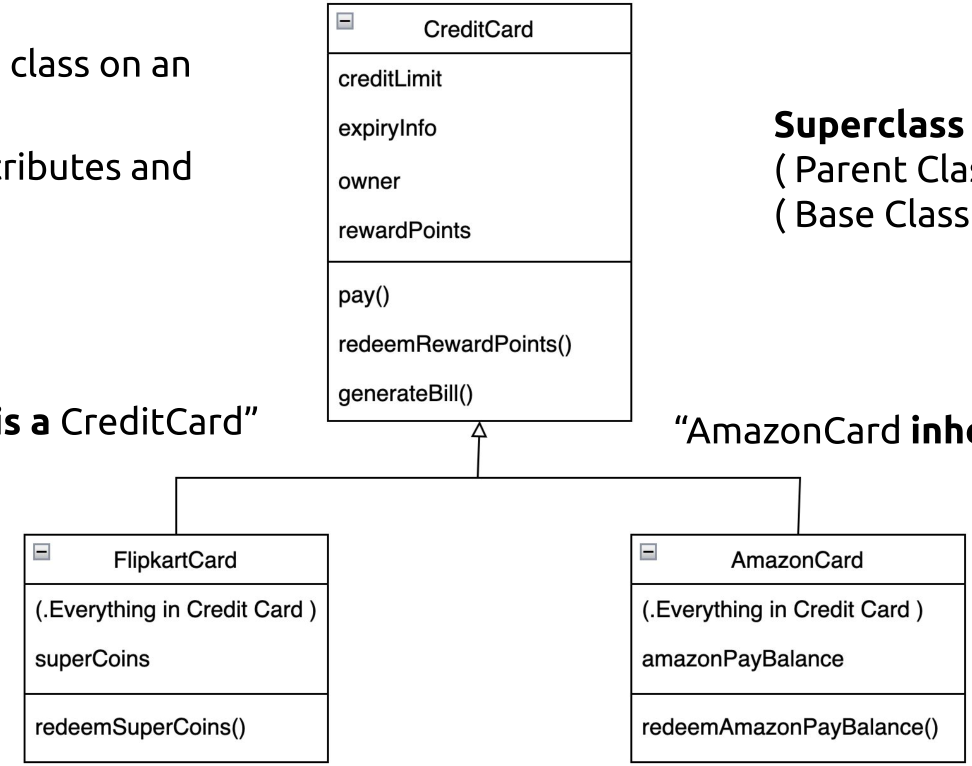
Why Inheritance?

- All credit cards have a **common set of features**.
- Some credit cards need to support **specific features.**
- Do we really need to implement all features from scratch for every new credit card type?
    - No, we can **avoid duplicate implementation of these features** across credit cards.
    - Example: SnapDeal can partner with one of the banks quickly for credit card.
    - Embraces **Reusability**
- Reduce development cost and time.

- How do we achieve this in software?
    - We can use **classes and inheritance**.

What is inheritance?

- Base a new object or a class on an existing one
- Inherit the existing attributes and methods
- **IS - A** Relationship



Superclass
(Parent Class)
(Base Class)

“FlipkartCard **is a** CreditCard”

“AmazonCard **inherits from** CreditCard”

Subclass
(Child Class)
(Derived Class)

Can you think of other such Inheritance scenarios?

- Amazon account and Prime Account
- Feature phone and a Smartphone
- Same Car Model, base variants and higher variants
- YouTube and YouTube Premium Account
- Common Bank functionality and Specific Bank Account functionality

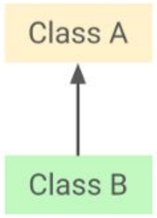
Activity #1 - Credit Card

- Clone this repository:

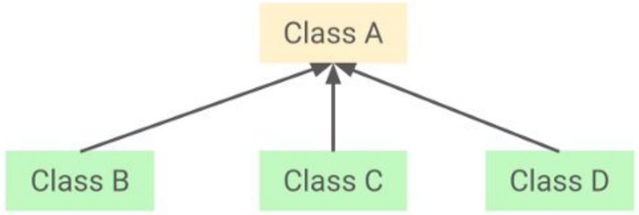
  https://gitlab.crio.do/public_content/bdt/session-activities/inheritance.git

- Open the folder: CrediCard-Java

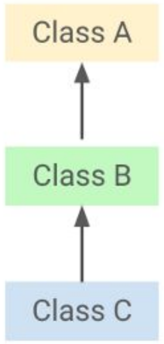
Types of Inheritance



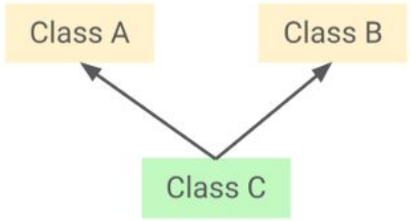
Single Inheritance



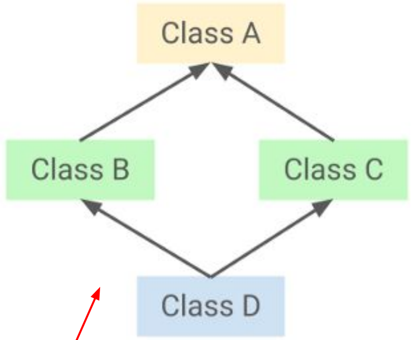
Hierarchical inheritance



Multilevel Inheritance



Multiple Inheritance



Hybrid Inheritance

Try out these inheritance code pieces from the cloned repository:

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance

Not supported in Java by **classes**. Why?
But possible using **interfaces** (Will be discussed later!)

Curious Cats

- Why is multiple inheritance not supported in Java with classes?
    - Let’s consider this scenario.
    - A, B, and C are three classes. The C class inherits A and B classes.
    - If A and B classes have an method with the same name and this method is invoked from C, which inherited method should be called? Method from A or Method from B?
    - Java shows up a compile-time error if you inherit 2 classes.

```
class A{
    void msg(){System.out.println("Hello");}
}

class B{
    void msg(){System.out.println("Welcome");}
}

class C extends A,B{

     public static void main(String args[]){
       C obj=new C();
       obj.msg(); //Now which msg() method
would be invoked?
}
```

Curious Cats

- Can constructors be inherited in Java?
  - A constructor cannot be inherited, as the subclasses always have a different name.
- What is the order in which constructors are invoked in Java?
  - Base class to derived class i.e. base class constructor gets invoked first when object of child class is created.
- Can we pass a child object to a method that is expecting the parent object as input parameter?
  - Yes
- Is there a specific syntax to invoke parent class method in child class?
  - No, the method can be simply invoked

5 minute break

protected keyword

- An access modifier that grants access of a class members to
  - Classes belonging to the same package as the given class

```
package p1;
public class Person {
    protected String name;
}
```

```
package p1;
public class Employer {
    void hireEmployee() {
        Person p = new Person();
        p.name = "Nam";    // access protected variable directly
    }
}
```

  - Subclass of the given class

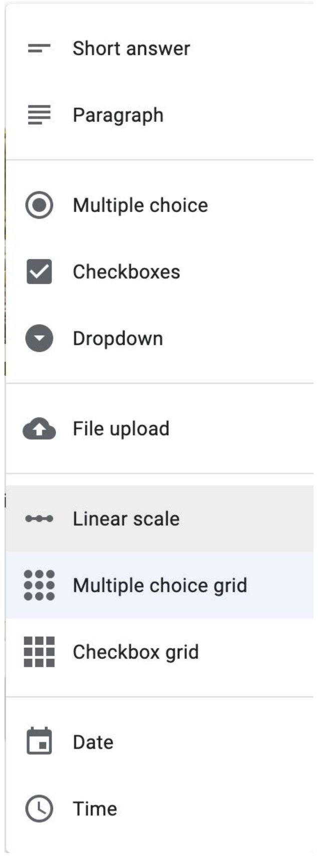
```
package p2;
import p1.Person;
class Employee extends Person {
    void doStuff() {
        Person p = new Person();
        p.name = "Bob";
    }
}
```

```
package p2;
import p1.Person;
class AnotherEmployer {
    void hire() {
        Person p = new Person();
        // compile error, cannot access protected variable
        // from different package
        p.name = "Nam";
    }
}
```

Activity 2 - Simple Quiz

- Create a class that represents a Quiz.
  - For example - Google form is a popular tool which allows you to create surveys, quiz, and much more.
- In this activity, we will be implementing a quiz application with these kind of questions:
  - Short Answer Questions
  - Multiple Choice Questions
- From the previously cloned repository, open the folder:

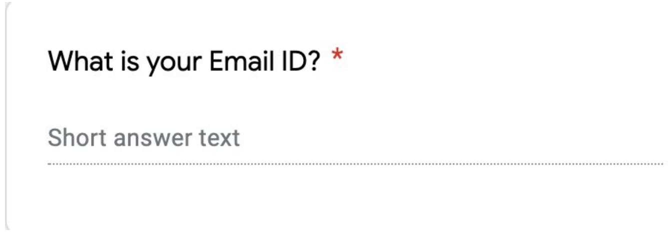
  GoogleFormClone-Java



Activity 2.1 - Short Answer Support

Short Answer Question

- What are the fields you can identify from this image?
  - question
  - answer
- What are the behaviours you think would be required for the fields?
  - Setters and getters
  - Check correct answer
  - Display the question
- Go to Activity 2.1 and implement the above defined requirements.
  - Instructions

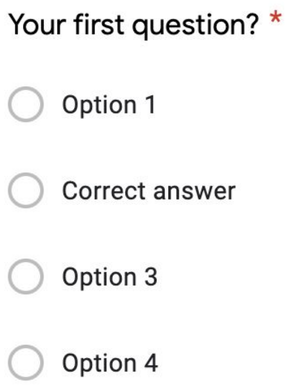

What is your Email ID? *

Short answer text

Activity 2.2 - Multiple Choice Question Support

Multiple Choice Question

- How does a Multiple Choice Question differ from Short Answer?
    - It store choices in addition to the question
- What are the extra fields you think might be required?
    - List of choices
- What are the extra behaviours you think would be required for the fields?
    - Add choices in the list
    - Display the MCQ question (Override)
- Is it possible to inherit the remaining fields/behaviour from Short Answer?
- Go to Activity 2.2 and implement the above defined requirements.
    - Instructions


Your first question? *

○ Option 1

○ Correct answer

○ Option 3

○ Option 4

Simple Quiz - New things we used

- **super keyword**
  - Refers to superclass (parent) objects.
- **protected access modifier**
  - An access modifier used for attributes, methods and constructors, making them accessible in the same package and subclasses.
- **Method overriding**
  - A child class can give its own implementation to a method which is already provided by the parent class.
  - In this case, when that method is invoked, the child class implementation will be used and NOT the parent class implementation.

Inheritance Exercises Byte Overview

Overview: Elementary Exercise - Google Form

[Let's Solve Elementary Exercise - Google Form]

Overview: Reinforcement Exercise - WhatsApp Message

Reinforcement Exercise - WhatsApp Message

Take home exercises for the session

- Inheritance Byte
  - Inheritance Quiz (Link Present in Byte)

These details are also available on the site.

Questions

1. What is inheritance in object-oriented programming? Provide an example.
2. What are the different types of inheritance in Java? Provide a brief explanation for each.
3. What is the significance of using the final keyword with methods and classes in Java?
4. Can Java support multiple inheritance? Explain.

Session Revision Quiz

Quiz Link

Solve this quiz to access your understanding of session's topics clearly

Week-1 Quiz

[Quiz Link]

Solve this quiz to access your understanding of all the session's topics you learnt this week

References

- Oracle Docs - Access Control

Further Reading

- Java Protected Keyword - Javatpoint

Thank you

Things to know about Java static methods

What will be the output?

```
class Calculate{
  private int x = 3;
  static int cube(){
    return x*x*x;
  }
  public static void main(String args[]){
    int result = Calculate.cube();
    System.out.println(result);
  }
}
```

Static methods can't use non-static (instance) variables.

What will be the output?

```
class Calculate{
  private int x = 3;
  public int getX(){
    return x;
  }
  static int cube(){
    return getX()*getX()*getX();
  }
  public static void main(String args[]){
    int result = Calculate.cube();
    System.out.println(result);
  }
}
```

Static methods can't use non-static methods either!

Curious Cats

- When does memory for the static variable get allocated?
    - Static variables are initialized
        - when class is loaded.
        - before any object of that class is created.
        - before any static method of the class executes.

Curious Cats

- Why is Java main method is static?
  - Stack Overflow Answer
- A static method can't access a non-static variable. But can a non-static method access a static variable?
  - Of course. A non-static method in a class can always call a static method in the class or access a static variable of the class.
- Can we have a static class?
  - A class can be declared static only if it is a nested class.

Curious Cats

- Are static local variables (a variable with scope limited to function) allowed in Java?
    - Try executing the following code snippet.

```
class Main {
  public static void main(String args[]) {
    System.out.println(decrement());
  }

  static int decrement()
  {
    static int x= 10;
    return x--;
  }
}
```

- A static variable is a class variable (for whole class).
- Hence compiler does not allow static local variable.

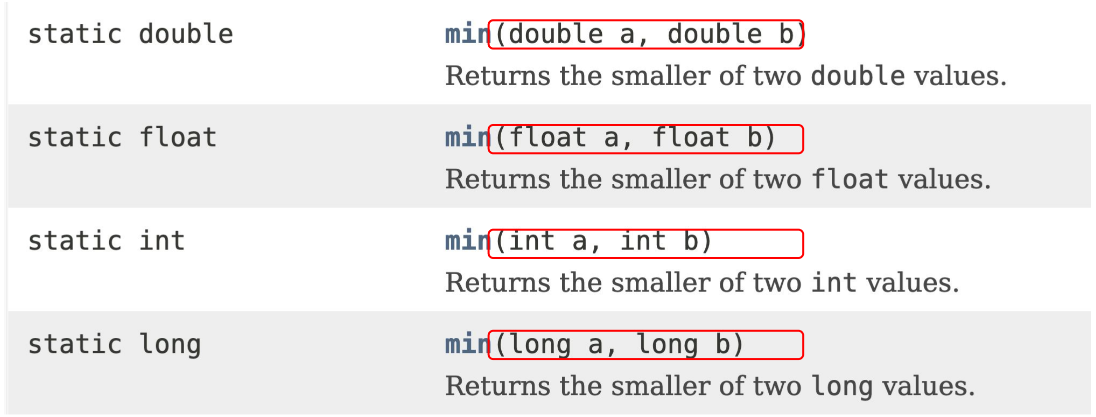
1. By Changing the number of arguments / parameters

```
class SimpleCalculator
{
   int add(int a, int b)
   {
       return a+b;
   }
   int  add(int a, int b, int c)
   {
       return a+b+c;
   }
}
public class Demo
{
  public static void main(String args[])
  {
     SimpleCalculator obj = new SimpleCalculator();
     System.out.println(obj.add(10, 20));
     System.out.println(obj.add(10, 20, 30));
  }
}
```

2. By Changing the Data Types of arguments

Find variations of Math.min()

- In Java's Math class, you will find many examples of overloaded methods.
- min() is overloaded with different data types.

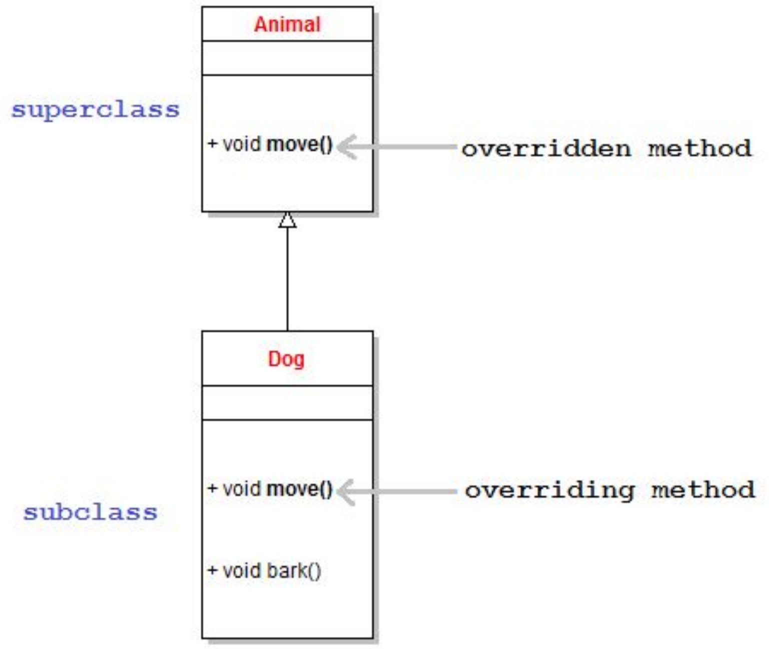


static double	min(double a, double b) Returns the smaller of two double values.
static float	min(float a, float b) Returns the smaller of two float values.
static int	min(int a, int b) Returns the smaller of two int values.
static long	min(long a, long b) Returns the smaller of two long values.

3. By changing the Order of Arguments

```
class Student
{
   public void show(String name, int age)
   {
      System.out.println("Name of person = "+name+ " and age is = "+ age);
   }
   public void show(int age, String name)
   {
      System.out.println("Name of person = "+name+ " and age is = "+ age);
   }
   public static void main (String [] args)
   {
     Student s = new Student();
     // If student providing parameter of String and int  type then first method called
     s.show("Ram", 25);
     // If student providing parameter of int and String type then second method called
     s.show(25, "Ram");
   }
}
```

Method Overriding



@Override notation

```
class Bank{
 //Overridden Method
 int getRateOfInterest(){return 5;}
}
//Creating child classes
class SBI extends Bank{
 //Overriding Method
 @Override
  int getRateOfInterest(){return 8;}
}
class ICICI extends Bank{
  //Overriding Method
 @Override
  int getRateOfInterest(){return 7;}
}

class Test{
 public static void main(String args[]){
   SBI s=new SBI();
   ICICI i=new ICICI();
   System.out.println("SBI Rate of Interest"+ s.getRateOfInterest());
   System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
 }
}
```

How to call an Overridden Method?

Suppose Base b = new Derived();

what is the result of the call b.methodOne();?

- A subclass might need to call the parent method for some operation to be successful.
- But the parent method is overridden, so how can we still call it?
- Use ***super.method()*** to force the parent's method to be called.

Curious Cats

- Can we overload main() method in Java?
    - Yes, but JVM calls that main() method that receives string array as an argument only.
- Try running the below code:

```
public class MainMethodOverloadingTest
{
        public static void main(String[] args)
        {
          System.out.println("main(String[] args)");
          main();
        }
       public static void main()
       {
         System.out.println("main without args");
        }
       public static void main(String args)
       {
         System.out.println("main with string args");
        }
}
```

Curious Cats

- **Can we override a static method?**
  - No, static methods cannot be overridden in Java.
  - Static methods are class-based and are called by class directly.
  - They don’t need objects to be invoked at runtime.
  - Hence the static method dispatch is determined by the compiler.
- **Can we override constructor?**
  - No, we cannot override a constructor.
  - Subclasses cannot override a parent class’s constructor as a constructor of two classes cannot be the same.
- **Do we really need to use @Override annotation?**
  - Not really but good to have.
  - Makes it human readable to understand that the method is a overriding method.
  - It helps to catch bug at compile time with less effort.