Object Oriented Programming

By Watcharin Sarachai

Topic

- Inheritance
- Constructors and Inheritance

Inheritance

Inheritance

Inheritance allows a class (child class) to inherit the features (fields and methods) of another class (parent class). In Java, a class can only extend one other class.

Goals

Code Reusability

Code Readability

In Java

Parent (also Super or Base) Class
and Child (also Sub or Derived) Class

Syntax:

```
extends
```

Inheritance

- The process by which one class acquires the properties **(data members)** and **functionalities(methods)** of another class is called **inheritance**.

- **Java Inheritance (Subclass and Superclass)** In Java, it is possible to inherit attributes and methods from one class to another.

- We can group the "inheritance concept" into two categories:

  - **subclass** (**child class**) - the class that inherits from another class

  - **superclass** (**parent, super or Base class**) - the class being inherited from

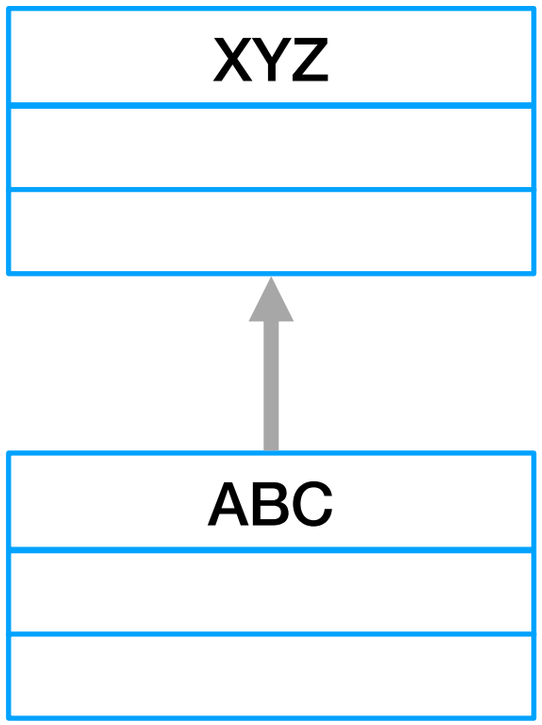
- To inherit from a class, use the **extends** keyword.

Syntax: Inheritance in Java

- To **inherit** a class we use extends keyword.
- Here class **XYZ** is **child** class and class **ABC** is **parent** class. The class **XYZ** is inheriting the properties and methods of **ABC** class.

```
class ABC extends XYZ
{
}
```

UML Diagram



Inheritance Example

- We have a base class **Teacher** and a sub class **PhysicsTeacher**.

- Since class **PhysicsTeacher** extends the designation and college properties and work() method from base class, we need not to declare these properties and method in sub class.

```
class Teacher {
  String designation = "Teacher";
  String collegeName = "Beginnersbook";

  void work() {
    System.out.println("Teaching");
  }
}

public class PhysicsTeacher extends Teacher {
  String mainSubject = "Physics";

  public static void main(String args[]) {
    PhysicsTeacher obj = new PhysicsTeacher();
    System.out.println(obj.collegeName);
    System.out.println(obj.designation);
    System.out.println(obj.mainSubject);
    obj.work();
  }
}
```

```
Output:

Beginnersbook
Teacher
Physics
Teaching
```

Inheritance Example

- Here we have **collegeName**, **designation** and work() method which are common to all the teachers so we have declared them in the **base class**, this way the child classes like **MathTeacher**, **MusicTeacher** and **PhysicsTeacher** do not need to write this code and can be used directly from base class.

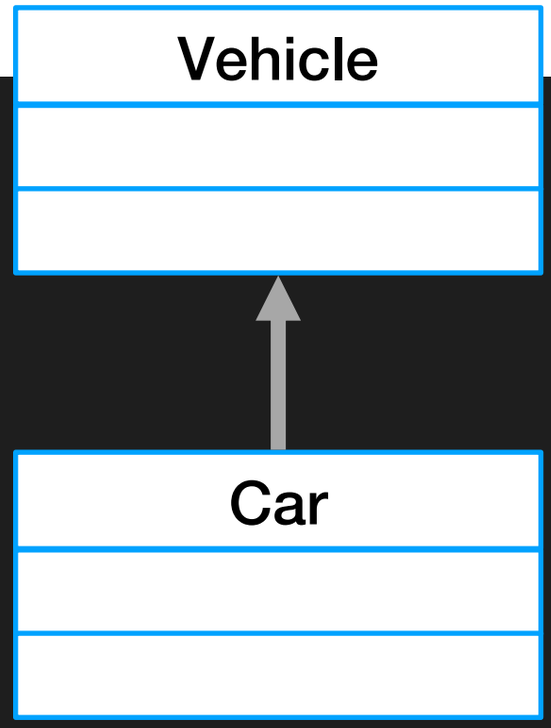
- Based on the above example we can say that **PhysicsTeacher** IS-A **Teacher**. This means that a child class has IS-A relationship with the parent class.

- This is inheritance is known as IS-A relationship between child and parent class

Inheritance Example

- In the example below, the Car class (subclass) inherits the attributes and methods from the Vehicle class (**superclass**):

UML Diagram




```
class Vehicle {
  protected String brand = "Ford";         // Vehicle attribute
  public void honk() {                     // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
  public static void main(String[] args) {

    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();

    // Display the value of the brand attribute (from the Vehicle class) and the value of the modelName
from the Car class
    System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

Inheritance and Encapsulation

- The derived class inherits all the members and methods that are declared as **public** or **protected**.

- If the members or methods of super class are declared as **private** then the derived class cannot use them directly. The private members can be accessed only in its own class.

- The private members can only be accessed using **public** or **protected getter and setter methods of super class** as shown in the slide example.

```java
class Teacher {
  private String designation = "Teacher";
  private String collegeName = "Beginnersbook";
  public String getDesignation() {
    return designation;
  }
  protected void setDesignation(String designation) {
    this.designation = designation;
  }
  protected String getCollegeName() {
    return collegeName;
  }
  protected void setCollegeName(String collegeName) {
    this.collegeName = collegeName;
  }
  void work() {
    System.out.println("Teaching");
  }
}
public class PhysicsTeacher extends Teacher {
  String mainSubject = "Physics";
  public static void main(String args[]) {
    PhysicsTeacher obj = new PhysicsTeacher();
    /*
     * Note: we are not accessing the data members directly we are using public
     * getter method to access the private members of parent class
     */
    System.out.println(obj.getCollegeName());
    System.out.println(obj.getDesignation());
    System.out.println(obj.mainSubject);
    obj.work();
  }
}
```

The output is:

```
Beginnersbook
Teacher
Physics
Teaching
```

Inheritance and Encapsulation

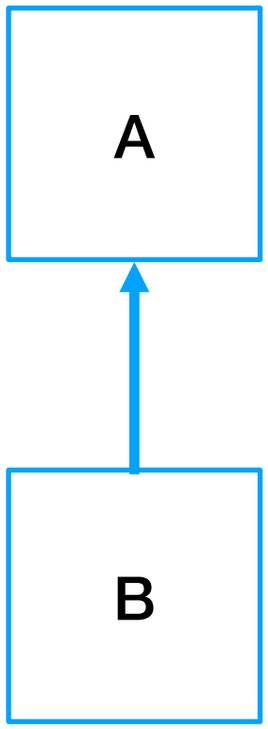
- The important point to note in the previous example is that the **child** class is able to access the private members of parent class through **protected** methods of parent class.

- When we make a instance variable (*data member*) or method protected, this means that they are accessible only in the **class itself** and in **child class**.

- These **public, protected, private** etc. are all access specifiers and we will discuss them in the coming next slide.

Types of inheritance

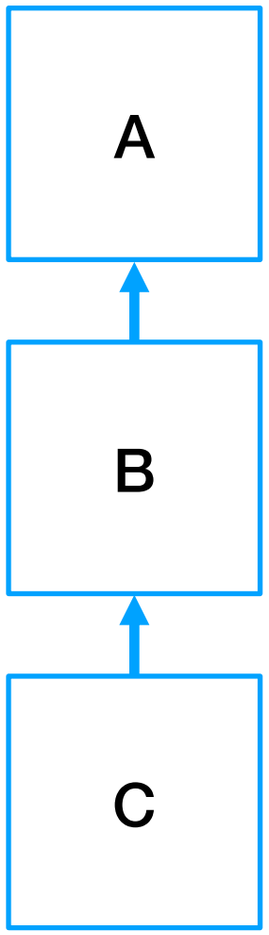
- **Single Inheritance:** refers to a **child** and **parent** class relationship where a class extends the another class



Single Inheritance

Types of inheritance

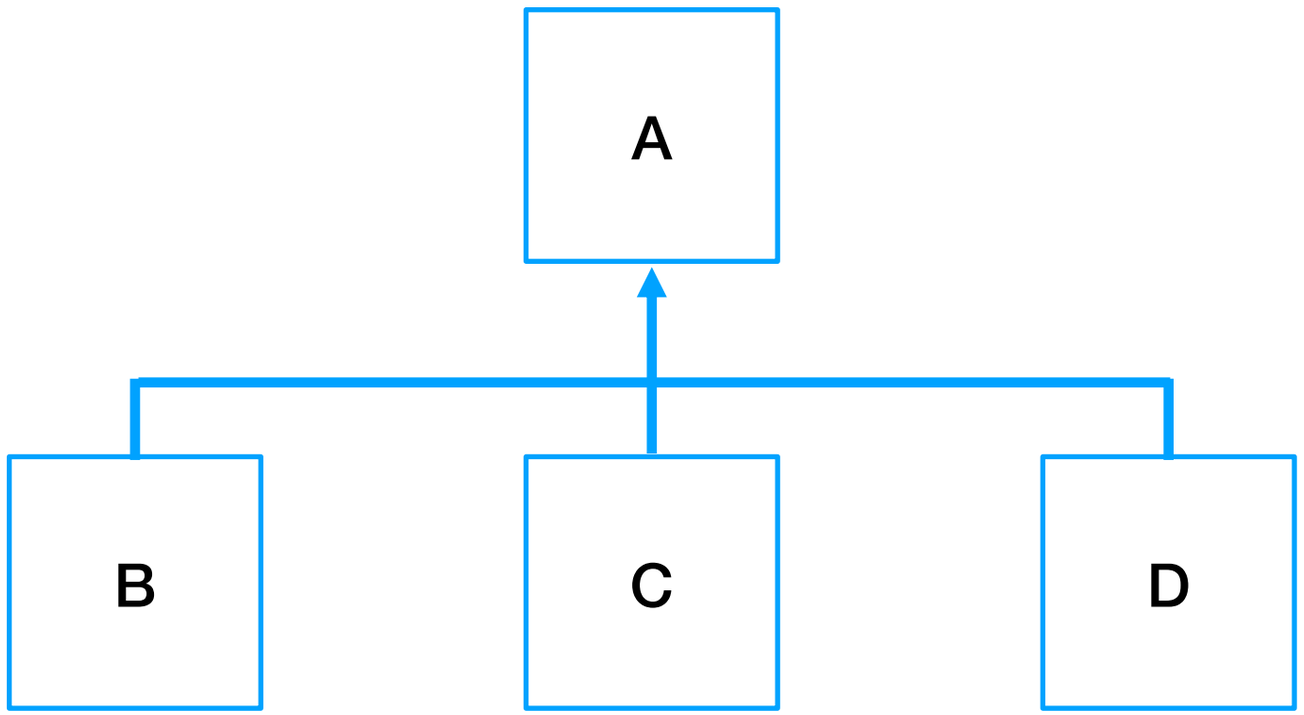
- **Multilevel inheritance:** refers to a child and parent class relationship where a class extends the child class. For example class C extends class B and class B extends class A.



Multilevel inheritance

Types of inheritance

- **Hierarchical inheritance:** refers to a child and parent class relationship where more than one classes extends the same class. For example, classes B, C & D extends the same class A.



Hierarchical inheritance

Constructors and Inheritance

Constructors and Inheritance

- **Constructor** of sub class is invoked when we create the object of **subclass**, it by default invokes the default constructor of **super** class.

- Hence, in inheritance the objects are constructed **top-down**. The superclass constructor can be called explicitly using the **super** keyword, but it should be first statement in a constructor.

- The **super** keyword refers to the **superclass**, immediately above of the calling class in the hierarchy.

Constructors and Inheritance

- The use of **super** keyword

  - To access the **data members** of parent class when both parent and child class have member with same name

  - To explicitly call the **no-arg** and **parameterized** constructor of parent class

  - To access the method of parent class when child class has **overridden that method**.

Constructors and Inheritance

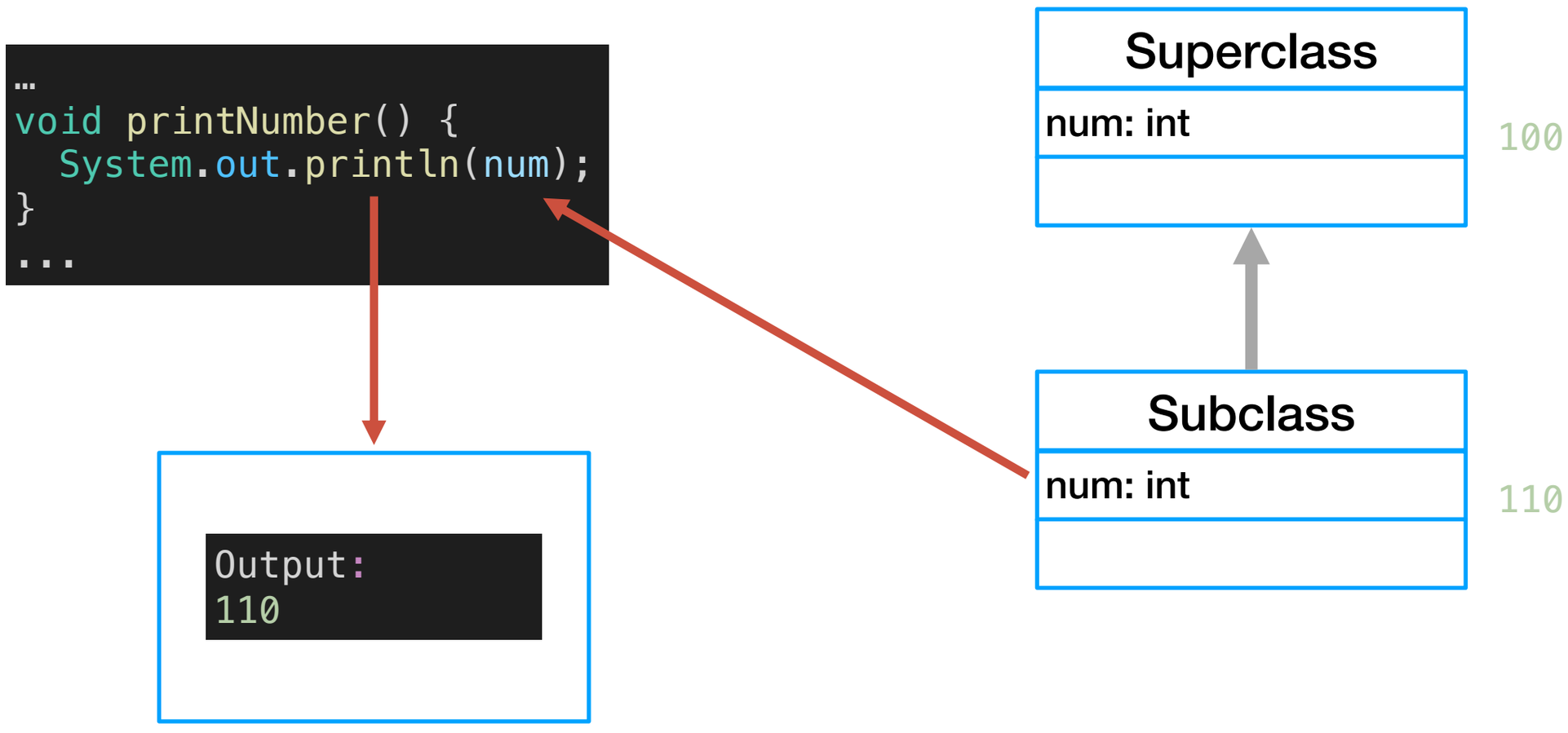
- How to use **super** keyword to access the variables of parent class

```java
//Parent class or Superclass or base class
class Superclass {
  int num = 100;
}

// Child class or subclass or derived class
class Subclass extends Superclass {
  /*
   * The same variable num is declared in the Subclass which is already present in
   * the Superclass
   */
  int num = 110;

  void printNumber() {
    System.out.println(num);
  }

  public static void main(String args[]) {
    Subclass obj = new Subclass();
    obj.printNumber();
  }
}
```

```
Output:

110
```

Constructors and Inheritance

- How to use **super** keyword to access the variables of parent class



Constructors and Inheritance

- Accessing the num variable of parent class:

  - By calling a variable like this, we can access the variable of parent class if both the classes (*parent and child*) have same variable.

```
super.variable_name
```

- Let’s take the same example that we have seen the previous slide, this time in print statement we are passing super.num instead of num.

Constructors and Inheritance

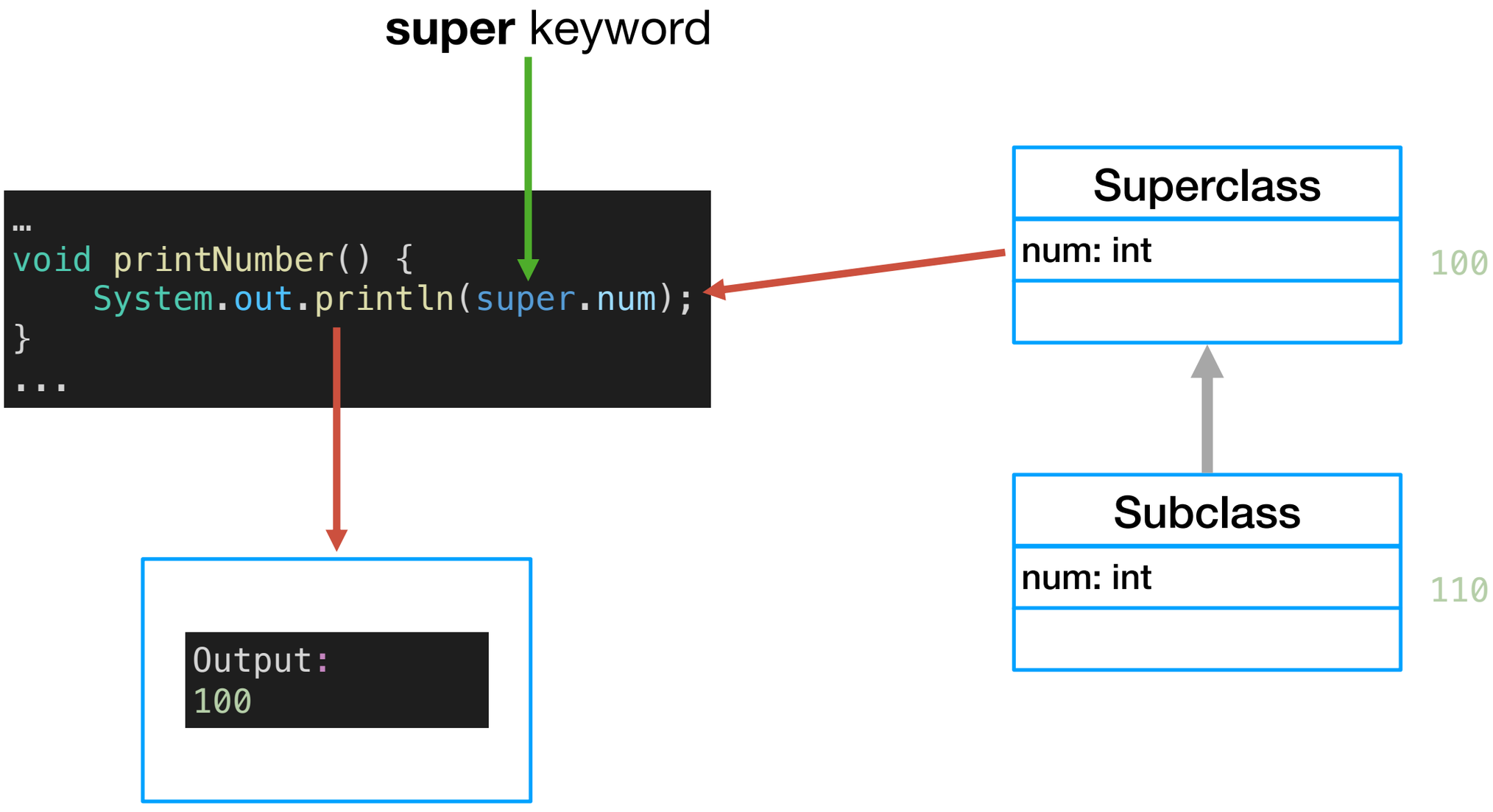
- How to use **super** keyword to access the variables of parent class

```java
//Parent class or Superclass or base class
class Superclass {
  int num = 100;
}

// Child class or subclass or derived class
class Subclass extends Superclass {
  /*
   * The same variable num is declared in the Subclass which is already present in
   * the Superclass
   */
  int num = 110;

  void printNumber() {
    System.out.println(super.num);
  }

  public static void main(String args[]) {
    Subclass obj = new Subclass();
    obj.printNumber();
  }
}
```

```
Output:

100
```

Constructors and Inheritance

- How to use **super** keyword to access the variables of parent class



Use of super keyword to invoke constructor of parent class

- When we create the object of sub class, the new keyword invokes the constructor of child class, which implicitly invokes the constructor of parent class.

- So the order to execution when we create the object of child class is:

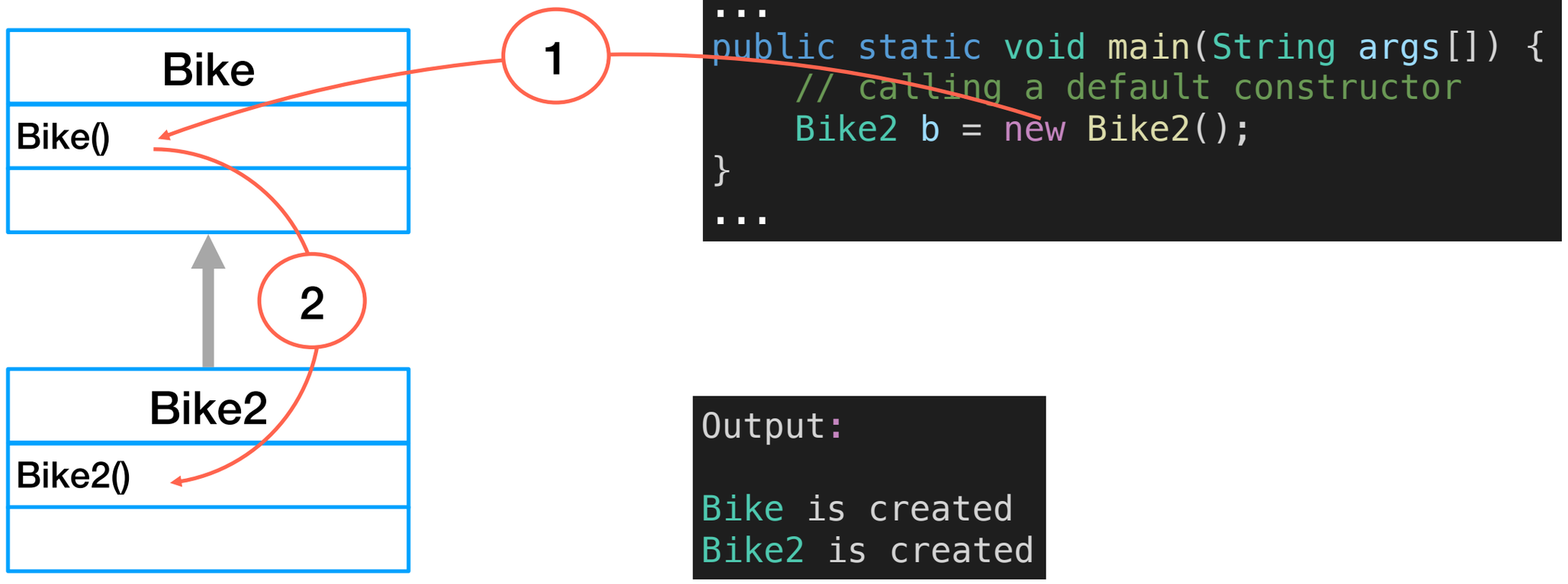
  1. Parent class constructor is executed first,

  2. Then the child class constructor is executed.

- It happens because compiler itself adds **super()** (*this invokes the no-arg constructor of parent class*) as the first statement in the constructor of child class.

Constructors and Inheritance

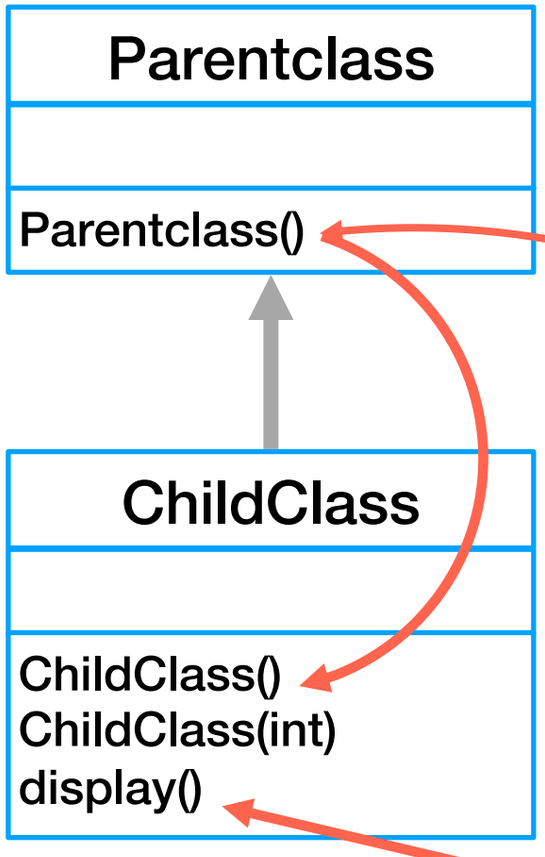
- Run Bike.java and Bike2.java in package camt.day3.inheritance and see the output.



```java
class Parentclass {
  Parentclass() { System.out.println("Constructor of parent class"); }
}
class ChildClass extends Parentclass {
  Subclass() {
    /*
    * Compile implicitly adds super() here as the first statement
    * of this constructor.
    */
    System.out.println("Constructor of child class");
  }
  Subclass(int num) {
    /*
    * Even though it is a parameterized constructor. The compiler
    * still adds the no-arg super() here
    */
    System.out.println("arg constructor of child class");
  }
  void display() { System.out.println("Hello!"); }
  public static void main(String args[]) {
    /*
    * Creating object using default constructor. This will invoke child class
    * constructor, which will invoke parent class constructor
    */
    ChildClass obj = new ChildClass();
    // Calling sub class method
    obj.display();
    /*
    * Creating second object using arg constructor it will invoke arg constructor
    * of child class which will invoke no-arg constructor of parent class
    * automatically
    */
    ChildClass obj = new ChildClass(10);
    obj2.display();
  }
}
```

```
Output:

Constructor of parent class
Constructor of child class
Hello!
Constructor of parent class
arg constructor of child class
Hello!
```

Constructors and Inheritance

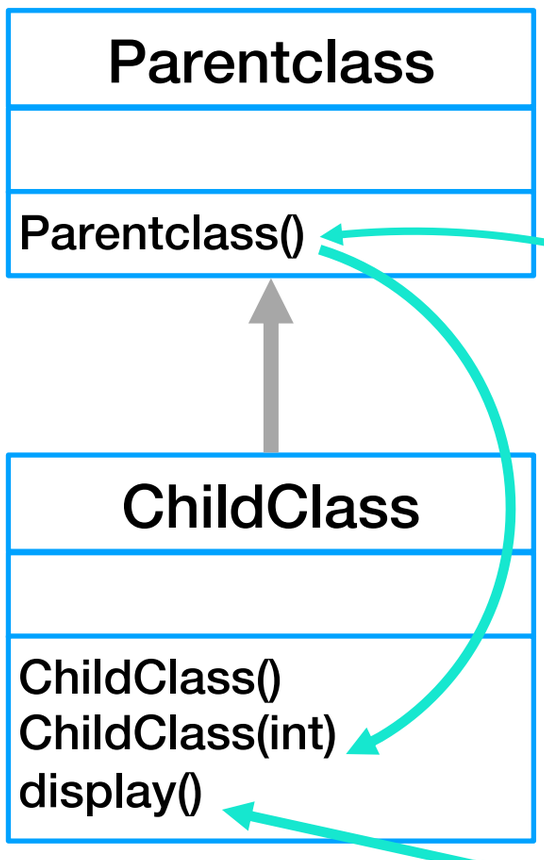



```
Subclass obj = new Subclass();

obj.display();
```

output

```
Output:

Constructor of parent class
Constructor of child class
Hello!


Constructor of parent class
arg constructor of child class
Hello!
```

Constructors and Inheritance




```
Subclass obj = new Subclass(10);

obj.display();
```

passed an argument 10

output

```
Output:

Constructor of parent class
Constructor of child class
Hello!
```

```
Constructor of parent class
arg constructor of child class
Hello!
```

Parameterized **super()** call to invoke parameterized constructor of parent class

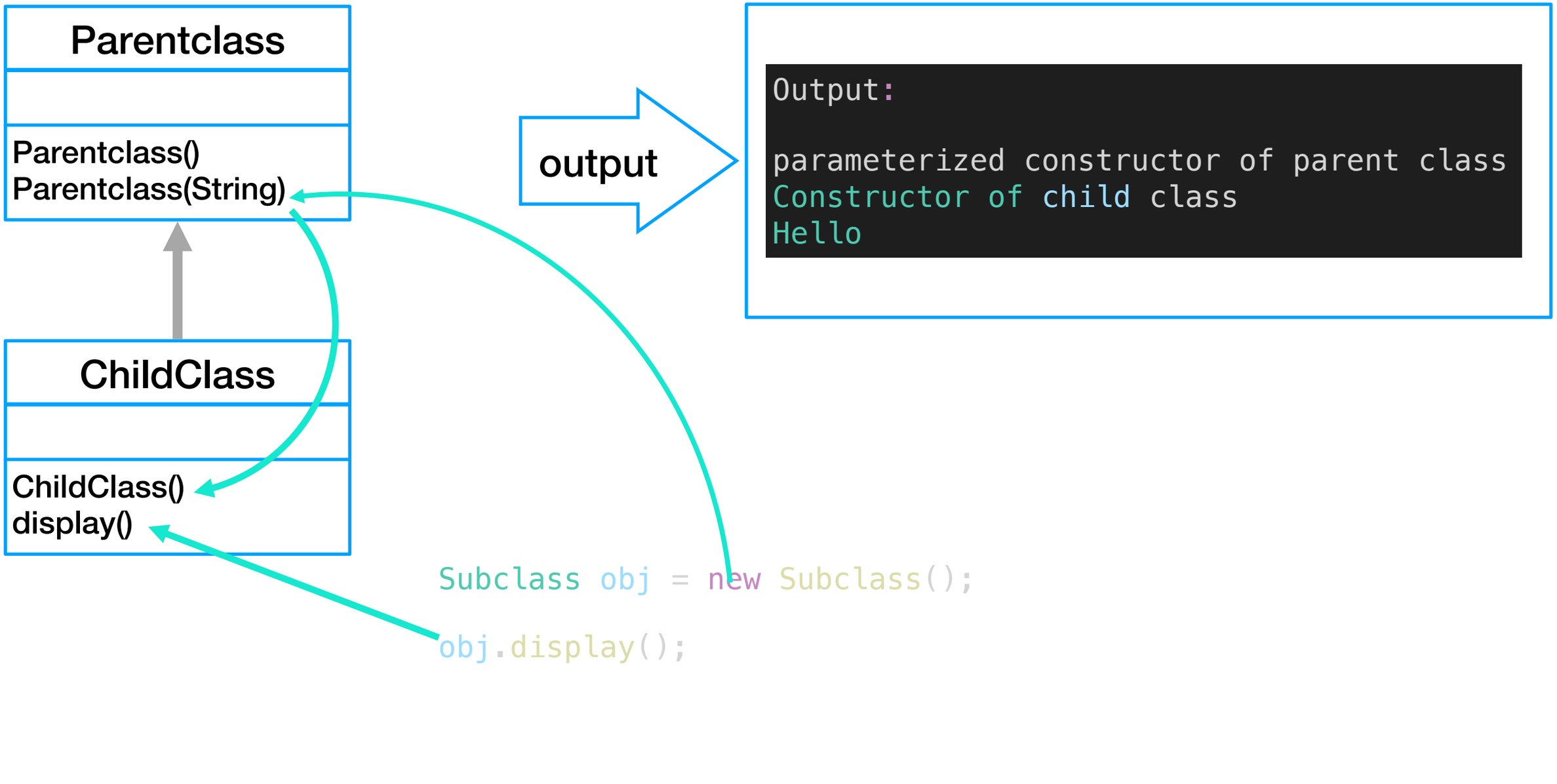
- We have a constructor in parent class that takes arguments then we can use parameterized super, like super(100); to invoke parameterized constructor of parent class from the constructor of child class.

- Let’s see an example to understand this:

```java
class Parentclass {
  // no-arg constructor
  Parentclass() {
    System.out.println("no-arg constructor of parent class");
  }

  // arg or parameterized constructor
  Parentclass(String str) {
    System.out.println("parameterized constructor of parent class");
  }
}

class ChildClass extends Parentclass {
  ChildClass() {
    /*
     * super() must be added to the first statement of constructor otherwise you
     * will get a compilation error. Another important point to note is that when we
     * explicitly use super in constructor the compiler doesn't invoke the parent
     * constructor automatically.
     */
    super("Hahaha");
    System.out.println("Constructor of child class");

  }

  void display() {
    System.out.println("Hello");
  }

  public static void main(String args[]) {
    ChildClass obj = new ChildClass();
    obj.display();
  }
}
```

```
Output:

parameterized constructor of parent class
Constructor of child class
Hello
```

Parameterized **super()** call to invoke parameterized constructor of parent class



Parameterized super() call to invoke parameterized constructor of parent class

- There are few important points to note in this example:

  1. **super()** or **parameterized** super must be the first statement in constructor otherwise we will get the compilation error: "*Constructor call must be the first statement in a constructor*"

  2. When we explicitly placed super in the constructor, the java compiler didn't call the default no-arg constructor of parent class.

Lab 05

- From the Lab03-EXERCISE-3:
    - Let class 'Square’ and make it inherit from the 'Rectangle' class with its constructor having a parameter for its side (suppose side) calling the constructor of its parent class as 'super(side, side)'. Print the area and perimeter to output.
    - Create a new class name “Shape” and define the “name” attribute to store the name of the shape, and the “numSide” attribute to store a number of shape sides.
    - Use the given ShapeRun class to test the program and modify it to display the Square detail. The example output is shown in the next slide.
    - This homework tests your knowledge of inheritance. Your solution every classes must inherit from Shape superclasses.

Lab 05

```
1.Circle
2.Rectangle
3.Triangle
4.Square
Please select [1-4]: 1
Enter the radius: 5
The circle area is : 78.54, and the perimeter is 31.42
```

```
1.Circle
2.Rectangle
3.Triangle
4.Square
Please select [1-4]: 2
Enter the length: 5
Enter the breadth: 7
The rectangle area is : 35.00, and the perimeter is 24.00
```

```
1.Circle
2.Rectangle
3.Triangle
4.Square
Please select [1-4]: 3
Enter the base: 3
Enter the height: 5
The triangle area is : 7.50, and the perimeter is 5.83
```

```
1.Circle
2.Rectangle
3.Triangle
4.Square
Please select [1-4]: 4
Enter the side: 5
The rectangle area is : 25.00, and the perimeter is 20.00
```

Reference

- **Super keyword in java with examplehttps://beginnersbook.com/2014/07/super-keyword-in-java-with-example/**
  Accessed: 2020-07-07
- **6 OOP Concepts in Java with examples [2020] · Raygun Bloghttps://raygun.com/blog/oop-concepts-java/**
  Accessed: 2020-07-04

```java
class ParentClass {
  // Parent class constructor
  ParentClass() {
    System.out.println("Constructor of Parent");
  }
}

class JavaExample extends ParentClass {
  JavaExample() {
    /*
     * It by default invokes the constructor of parent class You can use super() to
     * call the constructor of parent. It should be the first statement in the child
     * class constructor, you can also call the parameterized constructor of parent
     * class by using super like this: super(10), now this will invoke the
     * parameterized constructor of int arg
     */
    System.out.println("Constructor of Child");
  }

  public static void main(String args[]) {
    // Creating the object of child class
    new JavaExample();
  }
}
```

```
Output:

Constructor of Parent
Constructor of Child
```