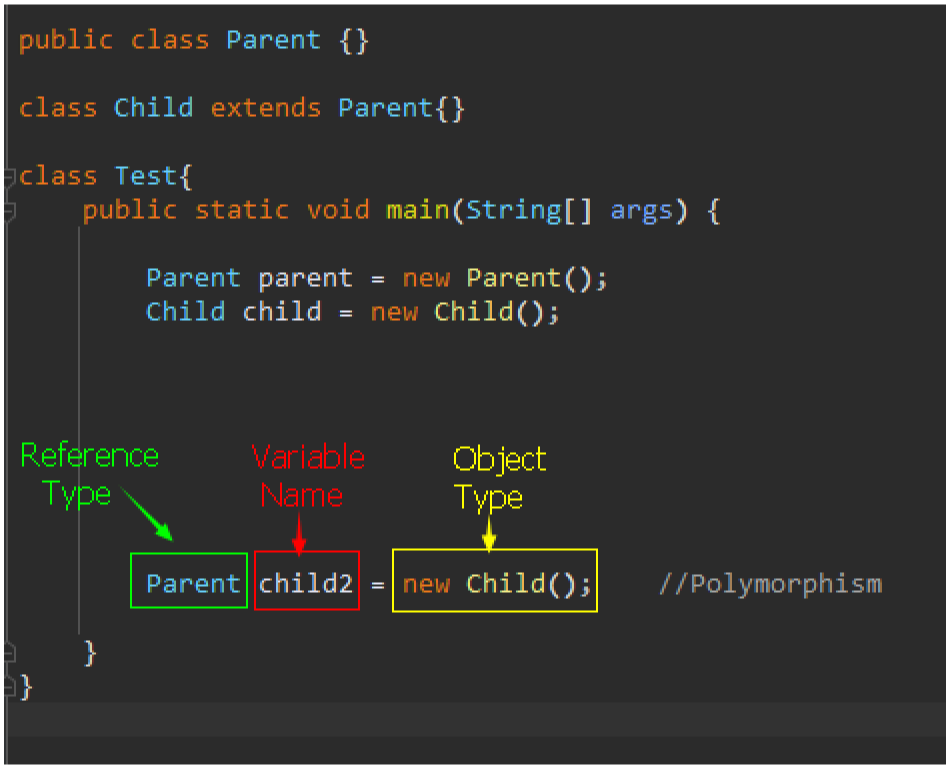
5-Polymorphism and Method/Variable Hiding Concept

▼ 1-In General

- Ability of an object to take many forms
- Polymorphism occurs when reference type is parent class/interface and object type is child

  Parent object = new Child()

```
//parent class
public class Parent {}

//child class
class Child extends Parent{}

class Test{
  public static void main(String[] args) {

    Parent parent = new Parent();
    Child child = new Child();

    Parent child2 = new Child();   //Polymorphic way

  }
}
```



```
public class Parent {}

class Child extends Parent{}

class Test{
    public static void main(String[] args) {

        Parent parent = new Parent();
        Child child = new Child();



        Parent child2 = new Child();      //Polymorphism

    }
}
```

- Polymorphism usage in our automation framework??

```
public class MyClass{

  @Test
  public void test1(){

     WebDriver driver;

     WebDriverManager.chromedriver().setup();
     driver = new ChromeDriver();

     WebDriverManager.edgedriver().setup();
     driver = new EdgeDriver();

     WebDriverManager.firefoxdriver().setup();
     driver = new FirefoxDriver();

  }

}
```

▼ isinstanceof operator

- instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface).

```
public class GrandFather {}

class Father extends GrandFather{}

class Child extends Father{}

class GrandChild extends Child{}


class Test{
  public static void main(String[] args) {

    GrandFather object = new Child();

    System.out.println(  object instanceof GrandFather);    //true
    System.out.println(  object instanceof Father);         //true
    System.out.println(  object instanceof Child);          //true
    System.out.println(  object instanceof GrandChild);     //false

  }
}
```

▼ 2-Polymorphism Types

- Compile Time Poly. (Static Poly) ⟶ Method Overloading
- Runtime Poly (Dynamic Poly) ⟶ Method Overriding

▼ 3-In polymorphic way, child class can only call the methods which parent class has

(Reference type decides what is accessible)

```
public class Parent {
  public void parentMethod(){
    System.out.println("parent method");
  }
}
```

```
class Child extends Parent{
  public void childMethod(){
    System.out.println("child method");
  }
}

class Test{
  public static void main(String[] args) {

    Child child = new Child();
    child.parentMethod();   //✔
    child.childMethod();    //✔

    Parent child2 = new Child();
    child2.parentMethod();  //✔

    child2.childMethod();   //✘       //!!!COMPILE ERROR!!!
    //reference type (Parent Class) does not have a method as childMethod()
    //That's why we can't call childMethod() in polymorphic way

    ((Child) child2).childMethod();   //✔
    //down casting, the way to access the method above
    //by down casting, we change the reference type to Child class itself
  }
}
```

```
public class Parent {

}

class Child extends Parent{
    public void childMethod(){
      System.out.println("child method");
    }
}

class Test{
    public static void main(String[] args) {

        Child child1 = new Child();
        child1.childMethod();               //✔

        Parent child2 = new Child();
        child2.childMethod();               //!!!COMPILE ERROR

        ((Child) child2).childMethod();     //✔ downcasting

  }
}

//-----------------------------------------------------------------
```

```
//child method
//!!!COMPILE ERROR
//child method
```

4-Variable Hiding

- Variable hiding happens when we define a variable in a child class with the same name as the one we defined in the parent class.
- (In simple terms: Parent Class and Child Class have the same variable)

```
public class Parent {
  String str = "Parent Class";
}

class Child extends Parent{
  String str = "Child Class";
}

class Test{
  public static void main(String[] args) {

    Parent parent = new Parent();
    System.out.println( parent.str );

    Child child1 = new Child();
    System.out.println( child1.str );

    Parent child2 = new Child();
    System.out.println( child2.str );

    System.out.println( ((Child) child2).str );   //downcasting

  }
}

//Parent Class
//Child Class
//Parent Class
//Child Class
```

```
public class Parent {
  String str = "Parent Class";
  String parentVariable = "Parent variable";
}
```

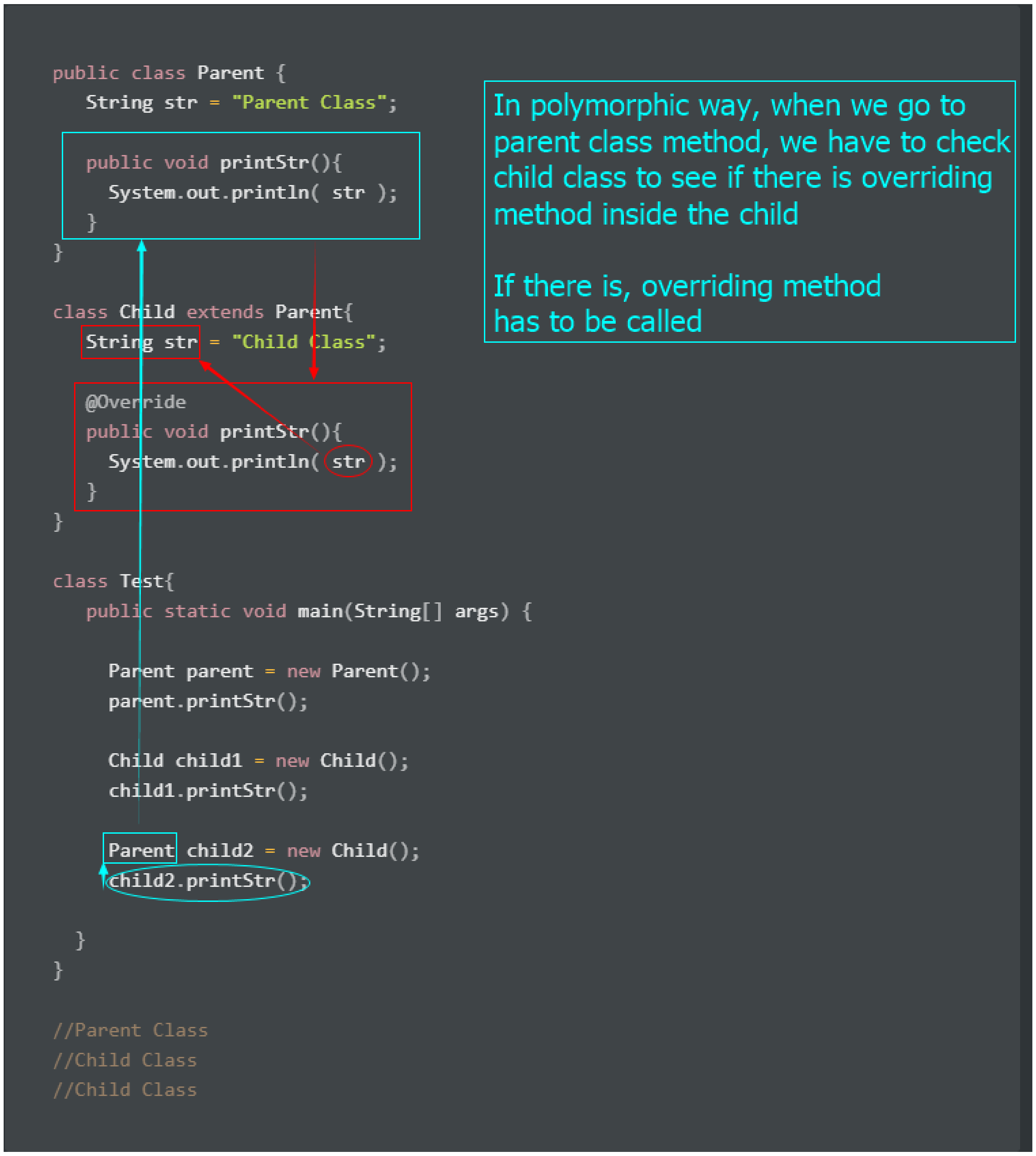
```
class Child extends Parent{
  String str = "Child Class";
  String childVariable = "Child variable";
}

class Test{
  public static void main(String[] args) {

    Parent parent = new Parent();
    System.out.println( parent.str );
    System.out.println( parent.parentVariable );

    Child child1 = new Child();
    System.out.println( child1.str );
    System.out.println( child1.childVariable );
    System.out.println( child1.parentVariable );

    Parent child2 = new Child();
    System.out.println( child2.str );
    System.out.println( child2.childVariable );   //!!!COMPILE ERROR!!!
    System.out.println( child2.parentVariable );


    System.out.println( ((Child) child2).str );   //downcasting

  }
}

//--------------------------------------------------------------------------
//Parent Class
//Parent variable
//Child Class
//Child variable
//Parent variable
//Parent Class
//!!!COMPILE ERROR!!!
//Parent variable
//Child Class
```

▼ 5-Method Hiding

- Method hiding follows exactly the same rules as method overriding
- Only difference is that, there is a static keyword in method declaration.

```
//No method hiding in this example, regular method overriding rules

public class Parent {
  String str = "Parent Class";

  public void printStr(){
    System.out.println( str );
  }
}

class Child extends Parent{
  String str = "Child Class";

  @Override
  public void printStr(){
    System.out.println( str );
  }
}

class Test{
  public static void main(String[] args) {

    Parent parent = new Parent();
    parent.printStr();

    Child child1 = new Child();
    child1.printStr();

//---------------------------------------

    Parent child2 = new Child();
    child2.printStr();

    ((Child) child2).printStr();       //downcasting

  }
}

//Parent Class
//Child Class
//Child Class
//Child Class
```

▼ click



```
public class Parent {
   String str = "Parent Class";

   public void printStr(){
     System.out.println( str );
   }
}

class Child extends Parent{
   String str = "Child Class";

   @Override
   public void printStr(){
     System.out.println( str );
   }
}

class Test{
   public static void main(String[] args) {

     Parent parent = new Parent();
     parent.printStr();

     Child child1 = new Child();
     child1.printStr();

     Parent child2 = new Child();
     child2.printStr();

   }
}

//Parent Class
//Child Class
//Child Class
```

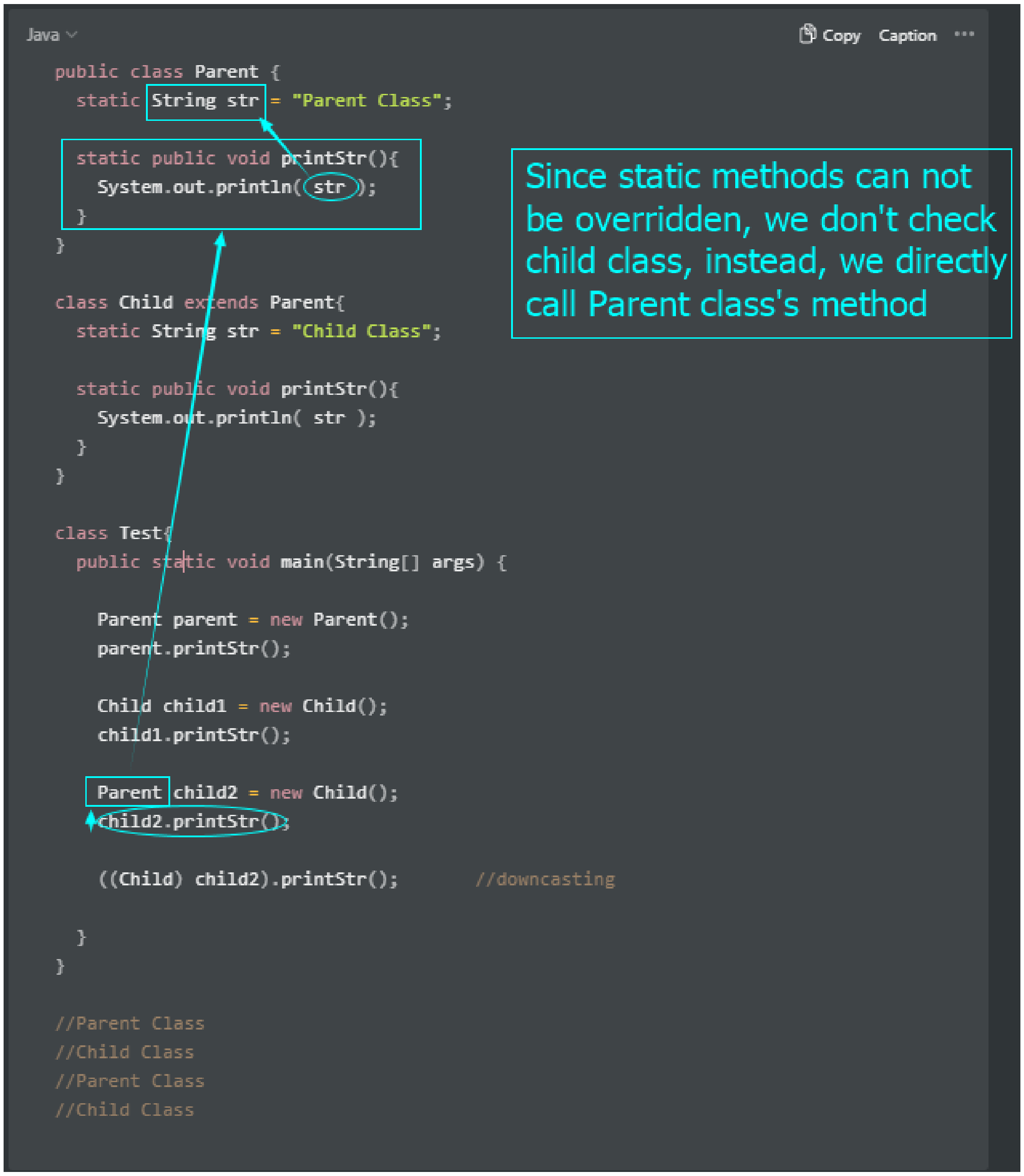
In polymorphic way, when we go to parent class method, we have to check child class to see if there is overriding method inside the child

If there is, overriding method has to be called

```
//There is method hiding in this example

public class Parent {
  static String str = "Parent Class";

  static public void printStr(){
    System.out.println( str );
  }
}
```

```
class Child extends Parent{
  static String str = "Child Class";

  static public void printStr(){
    System.out.println( str );
  }
}

class Test{
  public static void main(String[] args) {

    Parent.printStr();
    Child.printStr();

    Parent parent = new Parent();
    parent.printStr();

    Child child1 = new Child();
    child1.printStr();

//------------------------------------

    Parent child2 = new Child();
    child2.printStr();

    ((Child) child2).printStr();       //downcasting

  }
}

//Parent Class
//Child Class

//Parent Class

//Child Class

//Parent Class

//Child Class
```

▼ click


Java
Copy
Caption
Since static methods can not be overridden, we don't check child class, instead, we directly call Parent class's method

```java
public class Parent {
  static String str = "Parent Class";

  static public void printStr(){
    System.out.println( str );
  }
}

class Child extends Parent{
  static String str = "Child Class";

  static public void printStr(){
    System.out.println( str );
  }
}

class Test{
  public static void main(String[] args) {

    Parent parent = new Parent();
    parent.printStr();

    Child child1 = new Child();
    child1.printStr();

    Parent child2 = new Child();
    child2.printStr();

    ((Child) child2).printStr();       //downcasting

  }
}

//Parent Class
//Child Class
//Parent Class
//Child Class
```

- In method hiding, both methods have to be static, otherwise it will give a compiler error

```java
public class Parent {

  static public void method1(){
    System.out.println( "static method from parent" );
  }

}

class Child extends Parent{

  public void method1(){      //!!!COMPILE ERROR!!!
    System.out.println( "static method frm child" );
  }

}
```