Object Oriented Programming

By Watcharin Sarachai

Topic

- Overriding
- Polymorphism

Overriding

Method overriding

- Declaring a **method in sub class** which is already present in parent class is known as method overriding.
- **Overriding** is done so that a child class can give its own implementation to a method which is already provided by the parent class.
- In this case the method in parent class is called **overridden** method and the method in child class is called **overriding** method.

Method Overriding Example

- Lets take a simple example to understand this. We have two classes: A child class **Boy** and a parent class **Human**.

- The **Boy** class extends **Human** class. Both the classes have a common method void **eat()**.

- Boy class is giving its own implementation to the **eat()** method or in other words it is overriding the **eat()** method.

Method Overriding Example

- The purpose of Method Overriding: child class wants to give its own implementation so that when it calls this method, it prints Boy is eating instead of Human is eating.

```
class Human {
  // Overridden method
  public void eat() {
    System.out.println("Human is eating");
  }
}

class Boy extends Human {
  // Overriding method
  public void eat() {
    System.out.println("Boy is eating");
  }

  public static void main(String args[]) {
    Boy obj = new Boy();
    // This will call the child class version of eat()
    obj.eat();
  }
}
```

```
Output:

Boy is eating
```

Advantage of method overriding

- The main **advantage** of method **overriding** is that the class can **give its own specific implementation** to a inherited method without even modifying the parent class code.
- This is helpful when a class has several child classes.
- When a child class needs to use the parent class method, it can use it.
- However, if the other classes want to have different implementation, it can use overriding feature to make changes **without touching** the parent class code.

Method Overriding and Dynamic Method Dispatch

- Method Overriding is an example of runtime **polymorphism**.

- When a parent class reference points to the child class object then the call to the overridden method is determined at runtime.

- This because during method call which method (*parent class or child class*) is to be executed is determined by the type of object.

- The process that call to the overridden method is resolved at runtime is known as **dynamic method dispatch**.

- Lets see an example to understand this:

```java
class ABC {
  // Overridden method
  public void disp() {
    System.out.println("disp() method of parent class");
  }
}

class Demo extends ABC {
  // Overriding method
  public void disp() {
    System.out.println("disp() method of Child class");
  }

  public void newMethod() {
    System.out.println("new method of child class");
  }

  public static void main(String args[]) {
    /*
     * When Parent class reference refers to the parent class object then in this
     * case overridden method (the method of parent class) is called.
     */
    ABC obj = new ABC();
    obj.disp();

    /*
     * When parent class reference refers to the child class object then the
     * overriding method (method of child class) is called. This is called dynamic
     * method dispatch and runtime polymorphism
     */
    ABC obj2 = new Demo();
    obj2.disp();
  }
}
```

```
Output:

disp() method of parent class
disp() method of Child class
```

Method Overriding and Dynamic Method Dispatch

- In the previous example the call to the **disp()** method using second object (**obj2**) is runtime polymorphism (*or dynamic method dispatch*).

- Note: In dynamic method dispatch the object can call the overriding methods of child class and all the non-overridden methods of base class but it cannot call the methods which are newly declared in the child class.

- In the above example the object **obj2** is calling the **disp()**. However if you try to call the **newMethod()** method (*which has been newly declared in Demo class*) using **obj2** then you would give compilation error with the following message:

```
Exception in thread "main" java.lang.Error: Unresolved compilation
problem: The method xyz() is undefined for the type ABC
```

Rules of method overriding in Java

1. **Argument list:** The argument list of overriding method (*method of child class*) must match the Overridden method (*the method of parent class*). The data types of the arguments and their sequence should exactly match.

2. **Access Modifier** of the overriding method (*method of subclass*) cannot be more restrictive than the overridden method of parent class.

    - For e.g. if the **Access Modifier** of parent class method is public then the overriding method (*child class method*) cannot have private, protected and default Access modifier, because all of these three access modifiers are more restrictive than public.

Rules of method overriding in Java

```java
class MyBaseClass {
  public void disp() {
    System.out.println("Parent class method");
  }
}

class MyChildClass extends MyBaseClass {
  protected void disp() {
    System.out.println("Child class method");
  }

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

```
Output:

Exception in thread "main"
java.lang.Error: Unresolved
compilation
problem: Cannot reduce the
visibility of the inherited
method from MyBaseClass
```

Rules of method overriding in Java

```java
class MyBaseClass {
  protected void disp() {
    System.out.println("Parent class method");
  }
}

class MyChildClass extends MyBaseClass {
  public void disp() {
    System.out.println("Child class method");
  }

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

```
Output:

Child class method
```

However this is perfectly valid scenario as public is less restrictive than protected. Same access modifier is also a valid one.

Rules of method overriding in Java

3. **private**, **static** and **final** methods cannot be overridden as they are local to the class. However **static** methods can be re-declared in the sub class, in this case the sub-class method would act differently and will have nothing to do with the same static method of parent class.

4. Binding of overridden methods happen at runtime which is known as dynamic binding.

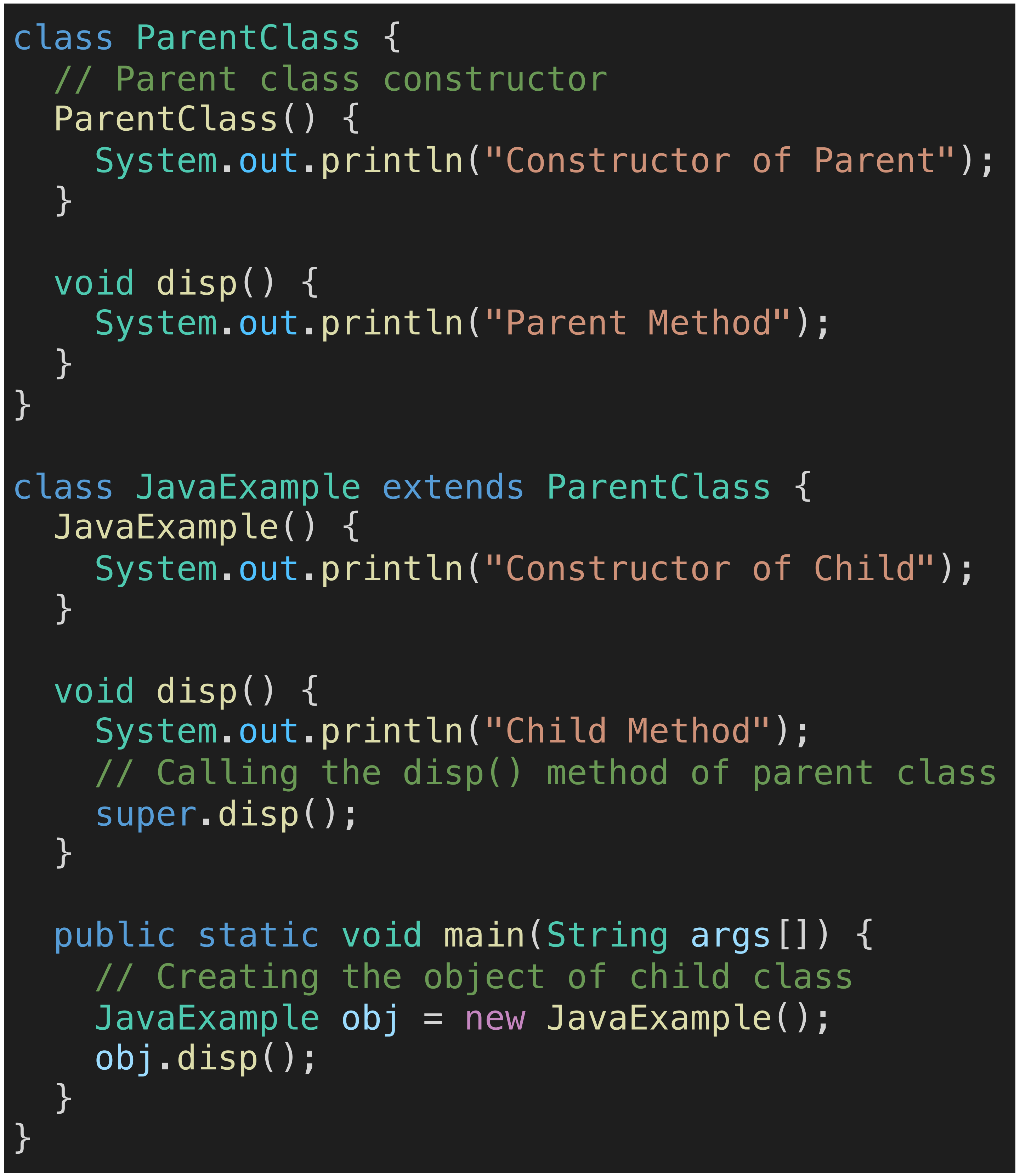
5. If a class is extending an abstract class or implementing an interface then it has to override all the abstract methods unless the class itself is a abstract class.

How to use super keyword in case of method overriding

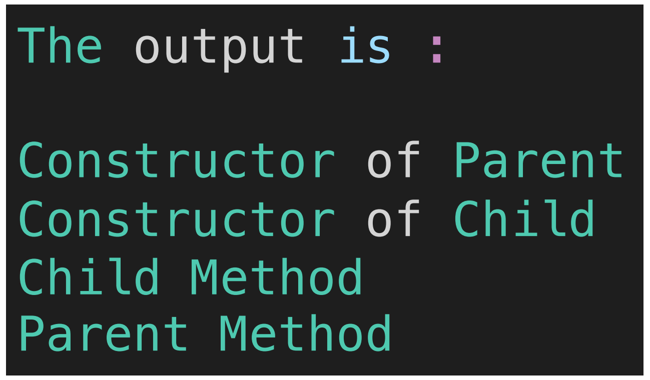
- When a child class declares a same method which is already present in the parent class then this is called method overriding.

- When a child class overrides a method of parent class, then the call to the method from child class object always call the child class version of the method.

- However by using super keyword like this: **super.method_name** we can call the method of parent class (the method which is overridden).



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

  void disp() {
    System.out.println("Parent Method");
  }
}

class JavaExample extends ParentClass {
  JavaExample() {
    System.out.println("Constructor of Child");
  }

  void disp() {
    System.out.println("Child Method");
    // Calling the disp() method of parent class
    super.disp();
  }

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



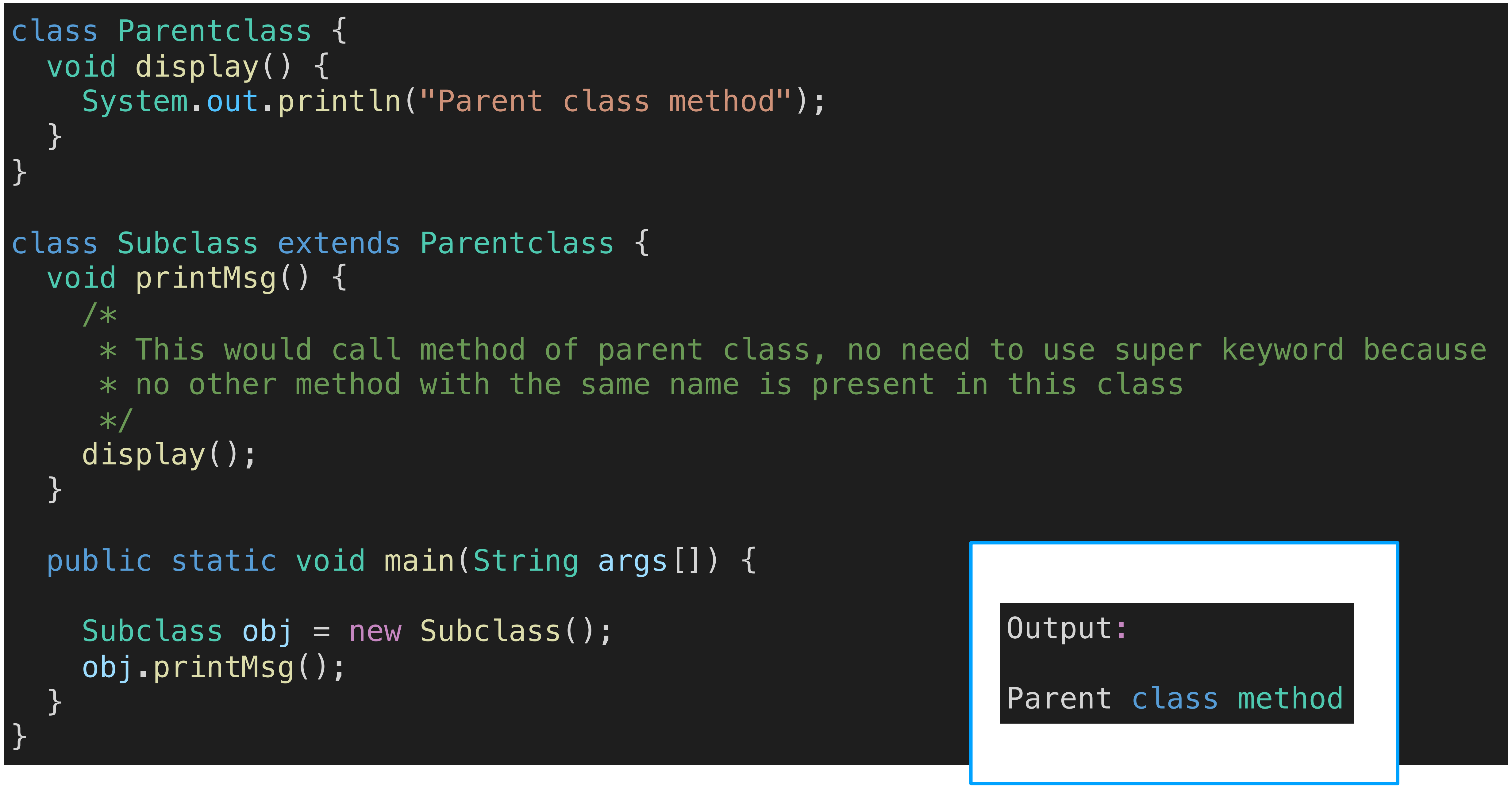
```
The output is :

Constructor of Parent
Constructor of Child
Child Method
Parent Method
```

```java
class Parentclass {
  // Overridden method
  void display() {
    System.out.println("Parent class method");
  }
}

class Subclass extends Parentclass {
  // Overriding method
  void display() {
    System.out.println("Child class method");
  }

  void printMsg() {
    // This would call Overriding method
    display();
    // This would call Overridden method
    super.display();
  }

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

```
Output:

Child class method
Parent class method
```

What if the child class is not overriding any method: No need of super

- When child class doesn’t override the parent class method then we don’t need to use the super keyword to call the parent class method.
- This is because in this case we have only one version of each method and child class has access to the parent class methods
- so we can directly call the methods of parent class without using super.



```java
class Parentclass {
  void display() {
    System.out.println("Parent class method");
  }
}

class Subclass extends Parentclass {
  void printMsg() {
    /*
     * This would call method of parent class, no need to use super keyword because
     * no other method with the same name is present in this class
     */
    display();
  }

  public static void main(String args[]) {

    Subclass obj = new Subclass();
    obj.printMsg();
  }
}
```

```
Output:

Parent class method
```

Polymorphism

Polymorphism

Polymorphism makes it possible to use the same code structure in different forms. Java classes can have various versions of the same method if their parameter structures are different.

Goals

- Better Implementing Inheritance
- Code Flexibility

In Java

Method Overloading (Static) and Method Overriding (Dynamic)

Syntax:

Static: myMethod() myMethod(int x) myMethod(int x, String y)

Dynamic: ParentClass.myMethod() ChildClass.myMethod()

Polymorphism

- **Polymorphism** is a object oriented programming feature that allows us to perform a single action in different ways.

- **For example,** lets say we have a class **Animal** that has a method **animalSound()**, here we cannot give implementation to this method.

- So, we make this method **abstract** like this:

```java
public abstract class Animal {
  ...
  public abstract void animalSound();
}
```

Polymorphism

- We have two **Animal** classes Dog and Lion that extends **Animal** class. We can provide the implementation detail there.

```
public class Lion extends Animal {
  ...
  @Override
  public void animalSound() {
    System.out.println("Roar");
  }
}
```

```
public class Dog extends Animal {
  ...
  @Override
  public void animalSound() {
    System.out.println("Woof");
  }
}
```

As you can see that although we had the common action for all subclasses animalSound() but there were different ways to do the same action. This is a perfect example of polymorphism (*feature that allows us to perform a single action in different ways*).

Types of Polymorphism

1. **Static Polymorphism**

   - **Polymorphism** that is resolved during compiler time is known as static polymorphism. Method overloading can be considered as static polymorphism example.

2. **Dynamic Polymorphism**

Types of Polymorphism

1.Static Polymorphism

```
class DisplayOverloading {
  public void disp(char c) {
    System.out.println(c);
  }

  public void disp(char c, int num) {
    System.out.println(c + " " + num);
  }
}

public class ExampleOverloading {
  public static void main(String args[]) {
    DisplayOverloading obj = new DisplayOverloading();
    obj.disp('a');
    obj.disp('a', 10);
  }
}
```

```
Output:

a
a 10
```

When we say method signature, we are not talking about return type of the method, for example if two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw compilation error.

Types of Polymorphism

2.Dynamic Polymorphism

- It is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime rather, thats why it is called runtime polymorphism.

```java
class Animal {
  public void animalSound() {
    System.out.println("Default Sound");
  }
}

public class Dog extends Animal {

  public void animalSound() {
    System.out.println("Woof");
  }

  public static void main(String args[]) {
    Animal obj = new Dog();
    obj.animalSound();
  }
}
```

```
Output:

Woof
```

Since both the classes, child class and parent class have the same method animalSound. Which of the method will be called is determined at runtime by JVM.

Types of Polymorphism

Few more overriding examples:

```java
Animal obj = new Animal();
obj.animalSound();
// This would call the Animal class method

Dog obj = new Dog();
obj.animalSound();
// This would call the Dog class method

Animal obj = new Dog();
obj.animalSound();
// This would call the Dog class method
```

Lab8

- From Lab7: Pinball game
  - Make a BackgroundPanel to accept a key press

  - Make a ball move within JPanel area.
    - If the ball hits the edge, the ball bounces back on the other direction.
    - Try to add more 100 balls.

Lab8

- From Lab7: Pinball game
  - Make a BackgroundPanel to accept a key press

```
public class BackgroundPanel extends JPanel implements ActionListener, KeyListener {
  …
  …

  @Override
  public void keyTyped(KeyEvent e) {}

  @Override
  public void keyPressed(KeyEvent e) {}

  @Override
  public void keyReleased(KeyEvent e) {
      System.out.println(e);
      char ch = e.getKeyChar();
      if (ch >= '0' && ch <= '5') {
          addBall(ch);
      }
      if (e.getKeyCode() == 8) { // backspace key
          removeBall();
      }
  }
}
```

Lab8

- Make a BackgroundPanel to accept a key press

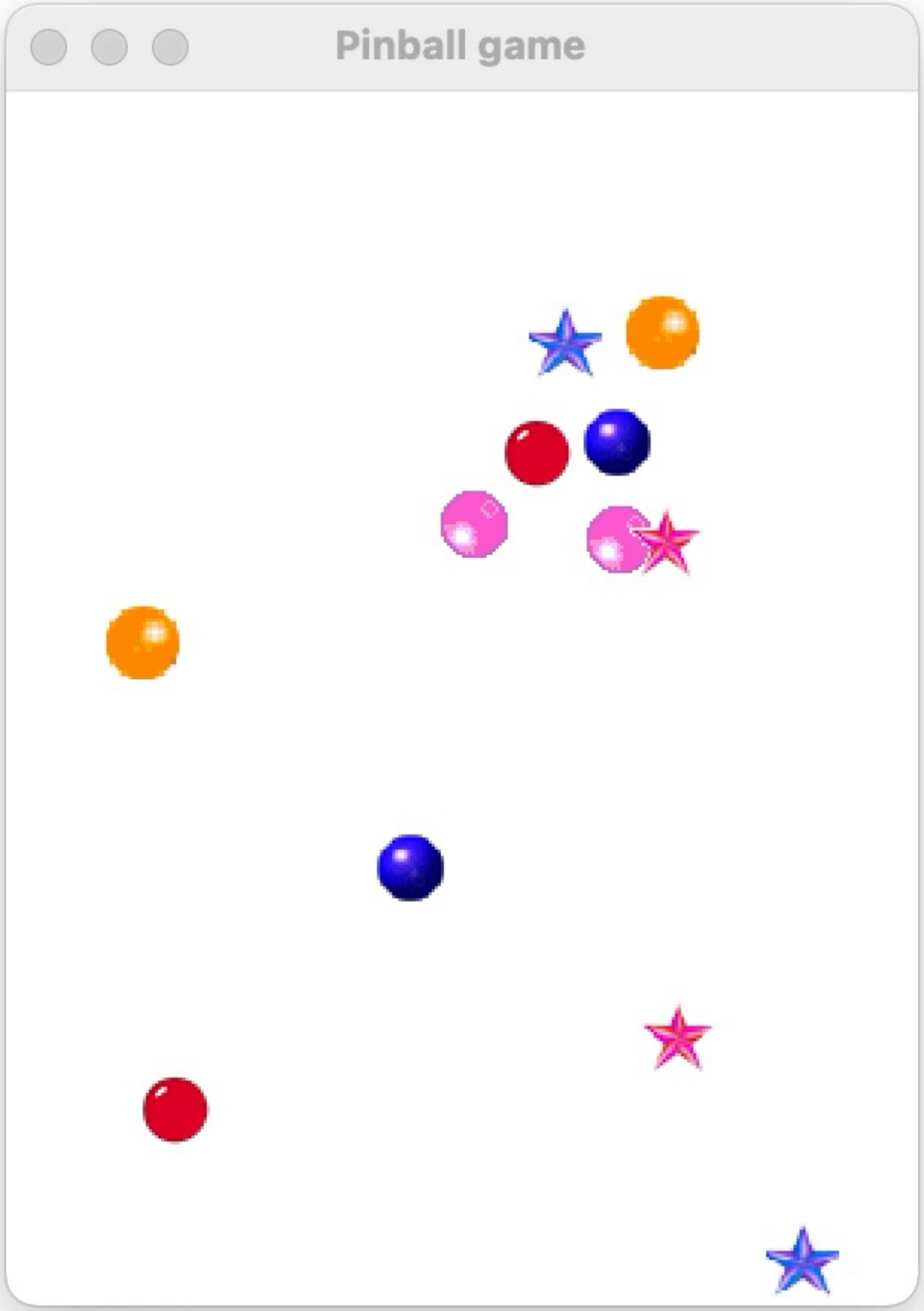
```
public class MainTest {
    public static void main(String [] args) {
        JFrame frame = new JFrame("Pinball game");
        BackgroundPanel panel = new BackgroundPanel();
        panel.addKeyListener(panel);
        panel.setFocusable(true);

        frame.add(panel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);

        Timer timer = new Timer(1000/32, panel);
        timer.start();
    }
}
```

Lab8

- Create subclass BlueBall, BlueStarBall, PinkBall, PinkStarBall, RedBall, and YellowBall, which inherit from Ball class. Each subclass displays a different image that is given from folder images.

- When pressing the keyboard 0, 1, 2, 3, 4, 5, the program will add the new ball into BackgroundPanel and bounce around the area.

- When pressing the backspace key the program will delete the first image in array.


Pinball game

Reference

- Super keyword in java with examplehttps://beginnersbook.com/2014/07/super-keyword-in-java-with-example/
  Accessed: 2020-07-07