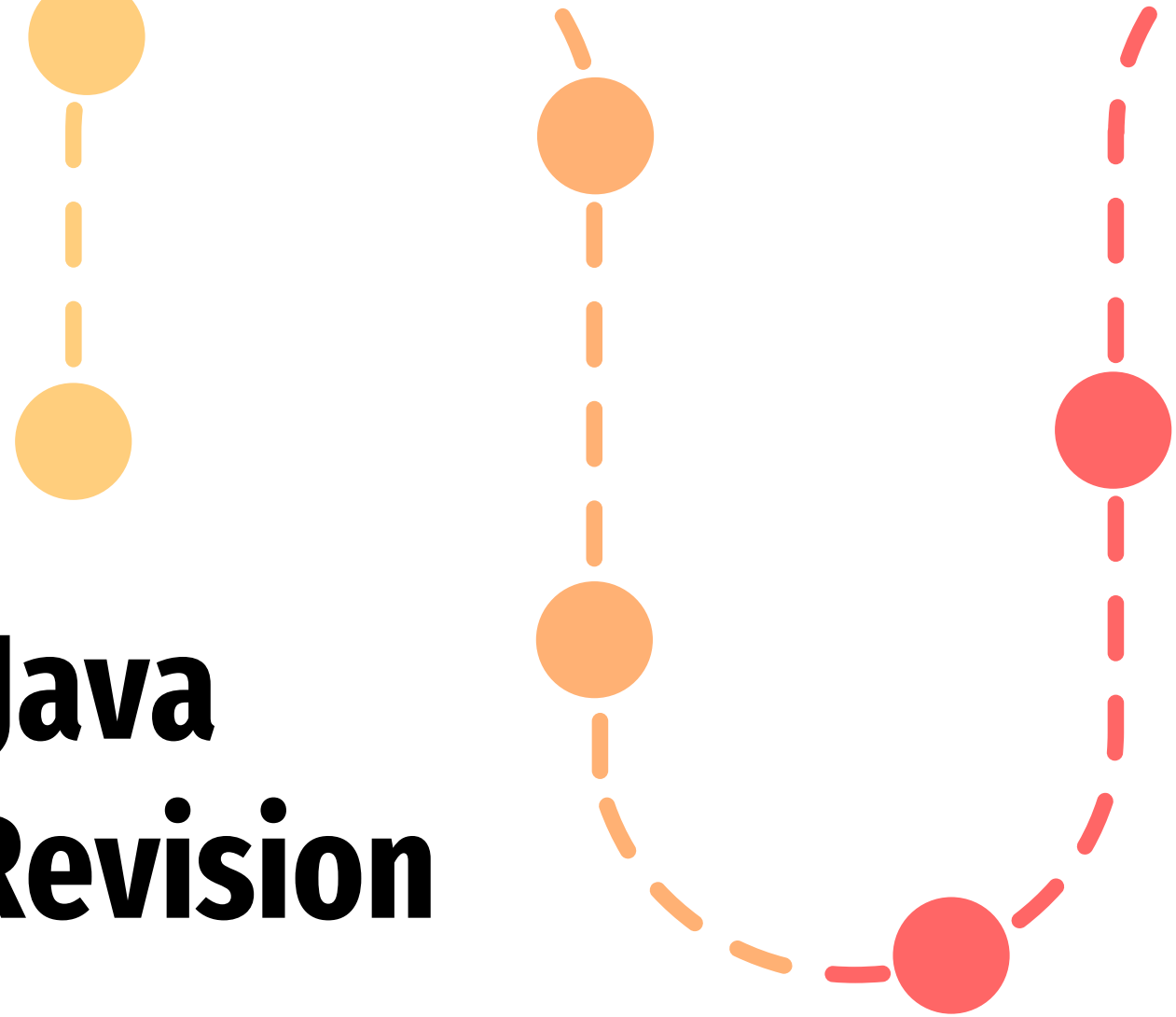


Advanced Java Concepts Revision

Amal Balabid – Associate Instructor

Abstraction

General concept

Abstract class

Interface

General concept

Abstraction is the methodology of hiding the implementation of internal details and showing the functionality to the users.

```
@GetMapping
public ResponseEntity<String> getIdAndName(@RequestParam String id, @RequestParam String name){
    return ResponseEntity.ok().body(id + " " + name);

}
```

Abstract class

Let’s say we are developing an application that accepts several types of orders: store orders, warehouse orders, and online orders.

Store Order

Warehouse Order

Online Order

Abstract class

```
abstract class Order {
    long order_id;
    List<Products> products;
    LocalDate order_date;


    abstract boolean validate();
    abstract void cancel();
    abstract String process();
}

class StoreOrder extends Order {
    Long store id
    String store_repo

    boolean validate(){
        //Custum implementation based on business logic
    }
    void cancel(){
        //Custum implementation based on business logic
    }

    String process(){
        //Custum implementation based on business logic
    }
}
```

Some common properties

Common methods

Own specific properties

Logic is different based on type

Interface

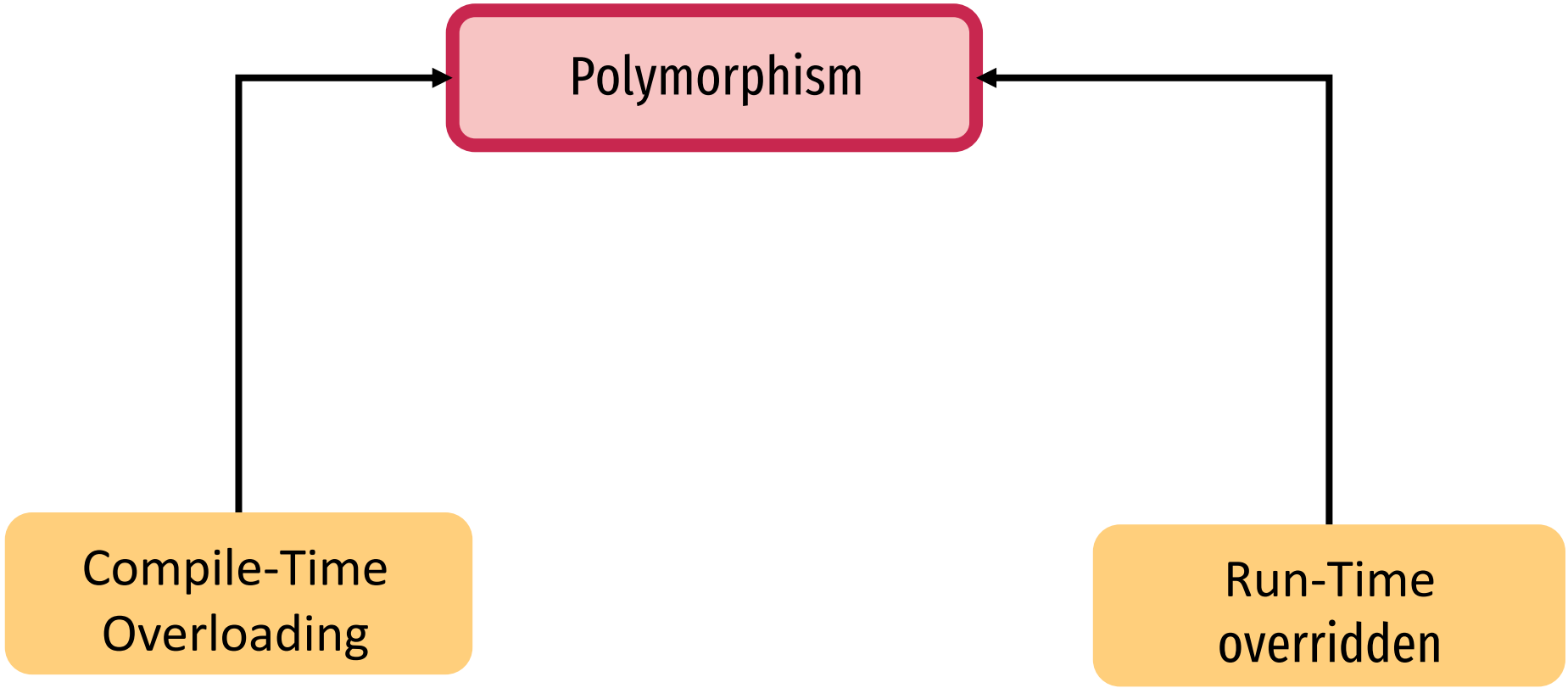
- It is used to achieve total abstraction.
- Support multiple inheritance in case of class (ex).

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    List<Product> findProductByName(String name);

}
```

Polymorphism

Many form

Polymorphism



Overloading

```java
public Product() {
}

public Product(Long id, String name, double price) {
    this.id = id;
    this.name = name;
    this.price = price;
}

Product p1 = new Product();
Product p2 = new Product( id: 1L,  name: "laptop",  price: 3000);
```

Overridden

```
  Returns: a string representation of the object.
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

```
@Override
public String toString() {
    return "Product{" +
            "id=" + id +
            ", name='" + name + '\'' +
            ", price=" + price +
            ", p1=" + p1 +
            ", p2=" + p2 +
            '}';
}
```

Collection

- List
- Queue
- Priority Queue
- Hash map
- Set

Queue

```
Queue<String> queue = new LinkedList<>();
queue.add("element 1"); //throws an exception in that case if the queue if full
queue.offer("element 2"); //returns false

queue.remove(); // throws an exception if the Queue is empty.
queue.poll(); // returns null if the Queue is empty


queue.element(); // throws a NoSuchElementException
queue.peek(); // returns null
```

P.Queue

```
Queue<Integer> pQueue= new PriorityQueue<Integer>();

pQueue.add(10);
pQueue.add(20);
pQueue.add(15);


System.out.println(pQueue.peek());

System.out.println(pQueue.poll());

System.out.println(pQueue.peek());
```

HashMap

```
HashMap<String, Integer> map = new HashMap<>();

map.put("a", 10);
map.put("b", 20);
map.put("c", 30);

System.out.println("Size of map is:- "
                   + map.size());

if (map.containsKey("a")) {
    Integer element = map.get("a");

    System.out.println("value for key"
                       + " \"a\" is:- " + element);
}
```

Thank you

You are welcome to ask any question