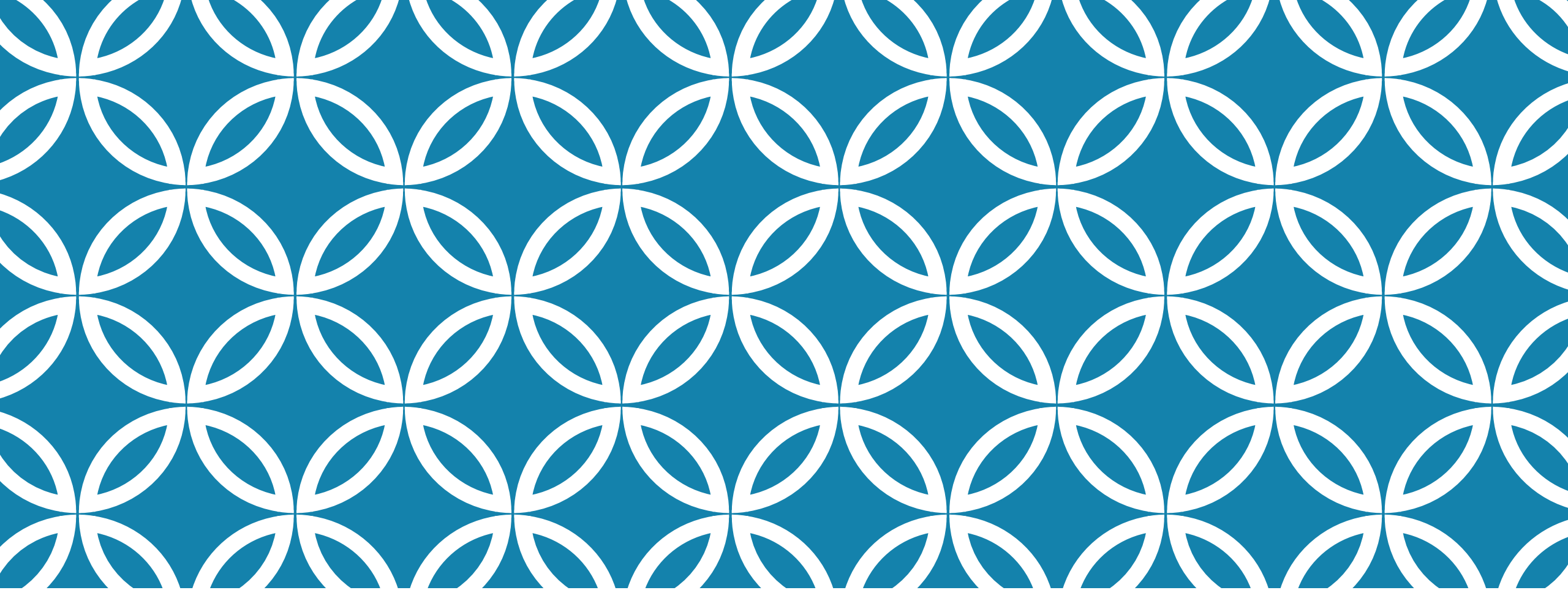


COMPUTER PROGRAMMING
WEEK 7

(MAR 13–17, 2023)

Instructor:

Abdul Aziz
Assistant Professor
(Computer Science Department)
National University- FAST (KHI Campus)

ACKNOWLEDGMENT

- Publish material by Virtual University of Pakistan.
- Publish material by Deitel & Deitel.
- Publish material by Robert Lafore.

THE SUBCLASS

- Inherits the data and methods of the parent class
- Does not inherit the constructors of the parent class

Opportunities:

1) add new data

2) add new methods

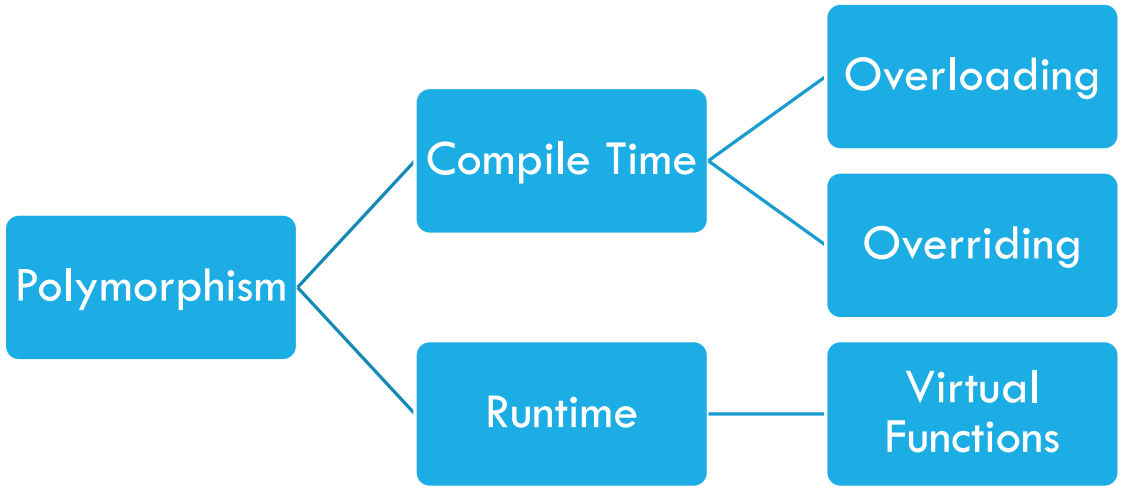
3) change the implementation of parent method.

INVOKING PARENT CLASS CONSTRUCTORS

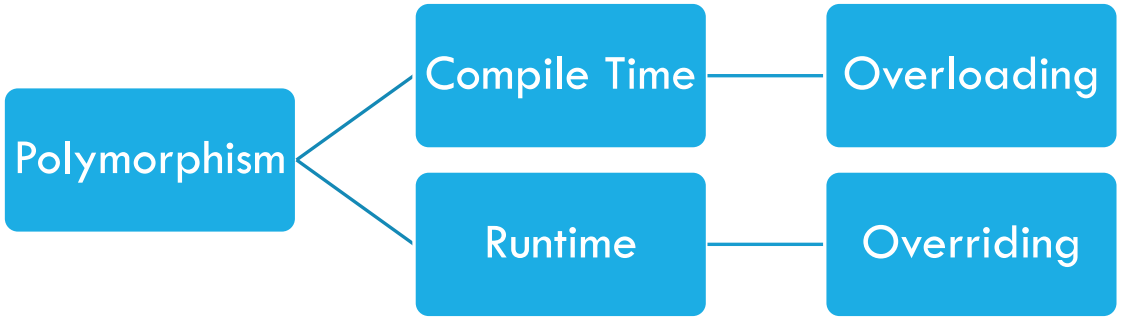
```
public  class Employee{
     protected String name;
     protected double salary;
     protected Date birthDate;
     public Employee( String name, double salary, Date birthDate){
          this.name = name;
          this.salary = salary;
          this.birthDate = birthDate;
     }
     //...
}
```

```
public  class Manager extends Employee{
     private String department;
     public Manager( String name, double salary, Date birthDate,
                                                  String department){
     super(name, salary, bityhDate);
          this.department = department;
     }
     //...
}
```

POLYMORPHISM



In Object Oriented Paradigm



In Java

OVERLOADING METHODS

Polymorphism: the ability to have many different forms

- Methods overloading:
  - methods having the **same name,**
  - argument list **must** differ,
  - return types **can be** different, but we cannot overload on return type
- Example:

```
public void println(int i)
public void println(float f)
public void println(String s)
```

OVERLOADING METHODS

The arguments can be differ in:

public void **println** (int i)

1- change number of arguments.

public void println(int i, string s, float f)

2- change type (data type) of arguments.

public void **println**(float i)

3- change position of arguments.

public void println(string s, float f, int i)

OVERRIDING METHODS

- A subclass can modify the **behavior** inherited from a parent class
- A subclass can create a method with different functionality than the parent's method but with the:
  - same **name**
  - same **argument list**
  - same **return type**

EXAMPLE

```
public  class Employee{
     protected String name;
     protected double salary;
     protected Date birthDate;
     public Employee( … ){
          // …
     }
     public String toString(){
          return "Name: "+name+" Salary: "+salary+" B. Date:"+birthDate;
     }
}
```

```
public  class Manager extends Employee{
     private String department;
     public Manager( … ){
          // …
     }

     public String toString(){
          return "Name: "+name+" Salary: "+salary+" B. Date:"+birthDate
                         +" department: "+department;
     }
}
```

INVOKING OVERRIDDEN METHODS

```
public  class Employee{
     protected String name;
     protected double salary;
     protected Date birthDate;
     public Employee( … ){
          // …
     }
     public String toString(){
          return "Name: "+name+" Salary: "+salary+" B. Date:"+birthDate;
     }
}
```

```
public  class Manager extends Employee{
     private String department;
     public Manager( … ){
          // …
     }
     public String toString(){
          return super.toString() + " department: "+department;
     }
}
```

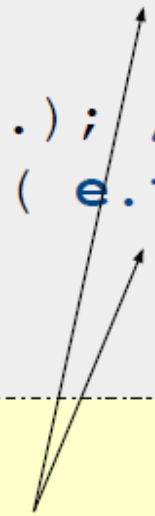
OVERRIDDEN METHODS CANNOT BE LESS ACCESSIBLE

```
public  class Parent{
    public void foo(){}
}

public class Child extends Parent{
    private void foo(){} //illegal
}
```

Overriding Methods

- *Polymorphism:* the ability to have many different forms

```
Employee e = new Employee(...);
System.out.println( e.toString() );

e = new Manager(...); //Correct
System.out.println( e.toString() );
```



Which `toString()` is invoked?

Static vs. Dynamic type of a reference

```
// static (compile time) type is: Employee
Employee e;

// dynamic (run time) type is: Employee
e = new Employee();

// dynamic (run time) type is: Manager
e = new Manager();
```

Polymorhic Arguments

```
public String createMessage( Employee e ){
   return "Hello, "+e.getName();
}

//...
Employee e1 = new Employee("Endre",2000,new Date(20,8, 1986));
Manager m1 = new Manager("Johann",3000,
                         new Date(15, 9, 1990),"Sales");

//...
System.out.println( createMessage( e1 ) );
System.out.println( createMessage( m1 ) );
```

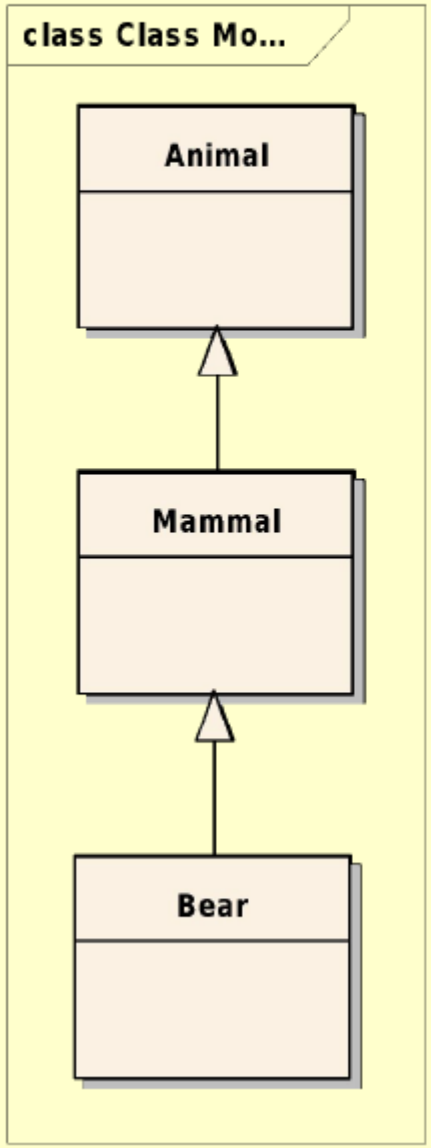
Heterogeneous Arrays

```
Employee emps[] = new Employee[ 100 ];
emps[ 0 ] = new Employee();
emps[ 1 ] = new Manager();
emps[ 2 ] = new Employee();
// …

// print employees
for( Employee e: emps ){
   System.out.println( e.toString() );
}

// count managers
int counter = 0;
for( Employee e: emps ){
   if( e instanceof Manager ){
       ++counter;
   }
}
```

Static vs. Dynamic type of a reference

```
Employee e = new Manager("Johann",3000,
                         new Date(10,9,1980),"sales");
System.out.println( e.getDepartment() );// ERROR

//Solution
System.out.println( ((Manager) e).getDepartment() );// CORRECT

//Better Solution
if( e instanceof Manager ){
     System.out.println( ((Manager) e).getDepartment() );
}
```

The `instanceof` Operator




```
Animal a = new Bear();

//expressions
a instanceof Animal  → true
a instanceof Mammal → true
a instanceof Bear → true
a instanceof Date → false
```