ERHVERVSAKADEMI
KØBENHAVN

Polymorfi

Polymorfi

- Polymorfi betyder "mange former"

- I objektorienteret programmering betyder det, at **forskellige objekter kan behandles ens**, selvom de er af forskellige typer — så længe de deler en fælles *supertype* (enten en fælles **superklasse** eller et **interface**).

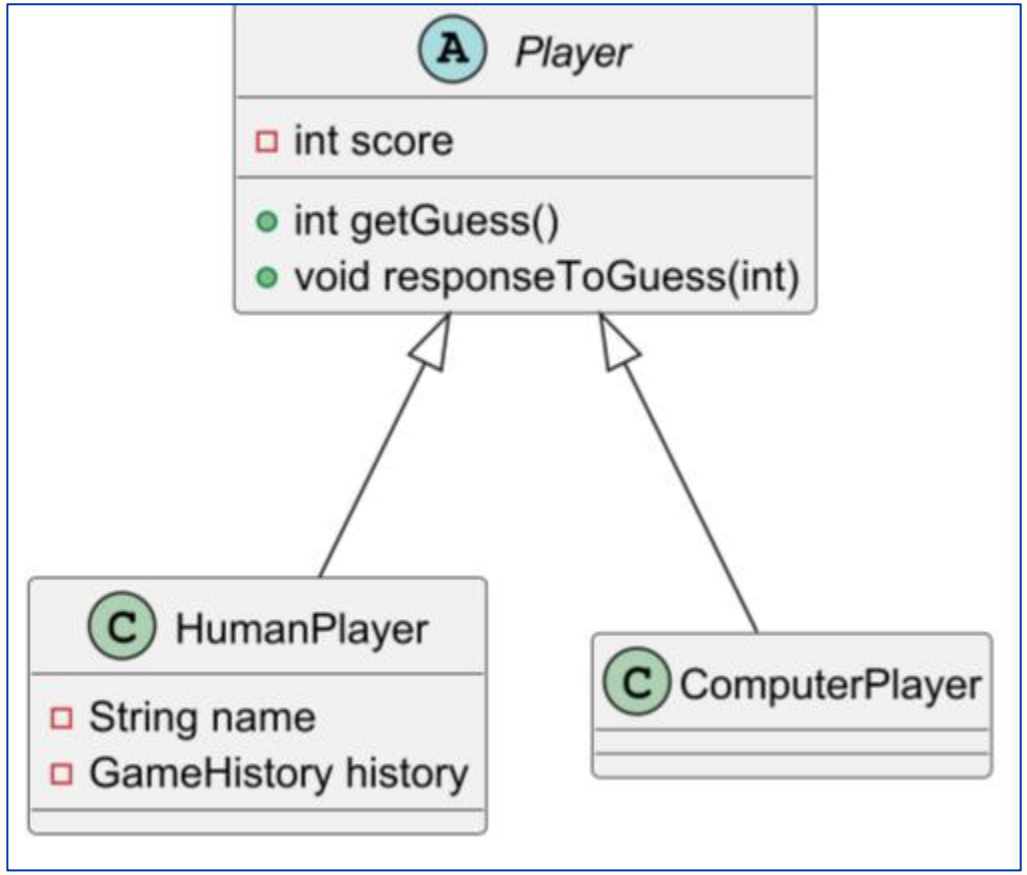
- Det betyder også at **ét objekt kan behandles forskelligt** alt efter hvilken type vores **reference** til den har.

Abstrakt klasse

- En **abstrakt klasse** er en *ufuldstændig* klasse, der kan indeholde:
  - **færdige metoder** (med kode)
  - **abstrakte metoder** (uden kode)

- Den kan **ikke instantieres** direkte med new.

```
Player player1 = new Player( score: 0);
Player player2 = new HumanPlayer( name: "Signe",  score: 0, io);
Player player3 = new SmartComputerPlayer();
```



Eksempel på abstrakt klasse og nedarving

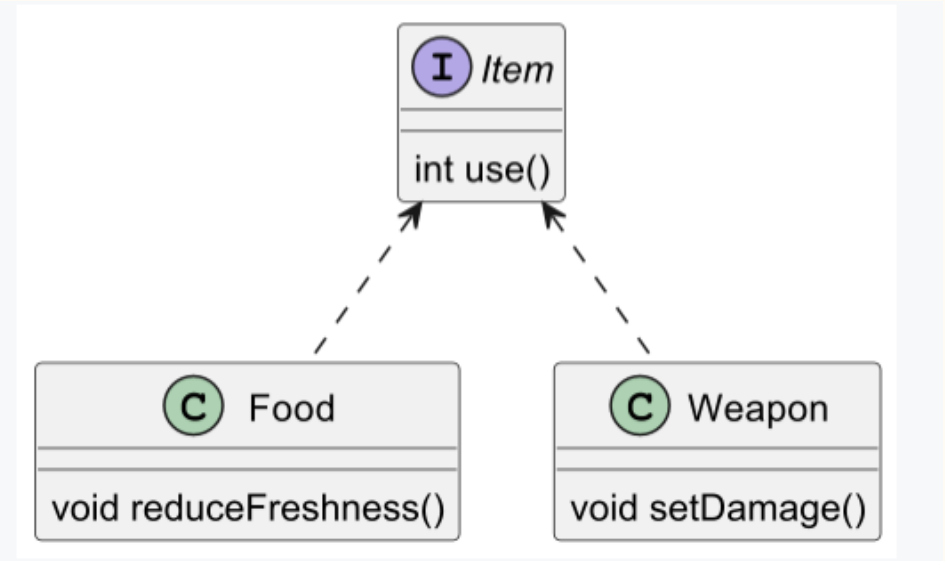
```java
public class HumanPlayer extends Player{
    private String name;
    private IO io;

    public HumanPlayer(String name, int score, IO io) {
        super(score);
        this.name = name;
        this.io = io;
    }

    @Override
    public int getGuess(){
        int result = io.promptInt( msg: "Indtast et tal");
        return result;
    }

    @Override
    public void responseToGuess(int response){
        if(response > 0)
            io.sendMessage( msg: "Desværre - gættet var for højt");
        else if (response < 0)
            io.sendMessage( msg: "Desværre - gættet var for lavt");
        else io.sendMessage( msg: "Tillykke " + name +
                    ". Du har gættet tallet!");
    }

    @Override
    public String getName() { return name; }

    @Override
    public boolean play(){
        String input = io.promptString( msg: "Vil du spille et nyt spil? y/n");
        if (input.equals("y")) return true;
        else return false;
    }
}
```

```java
public abstract class Player {  3 inheritors
    protected int score;

    public Player(int score) { this.score = score; }

    public boolean play() { return true; }

    public int getScore() { return score; }

    public void setScore(int score) {
        if(score < this.score)
            this.score = score;
    }

    public abstract int getGuess();  3 implementations

    public abstract void responseToGuess(int response);  2 implementations

    public abstract String getName();  3 implementations

    @Override
    public String toString() {
        return "Player{" +
                ", score=" + score +
                '}';
    }
}
```

```java
public class SmartComputerPlayer extends Player{
    int min;
    int max;
    int previousGuess;

    public SmartComputerPlayer(){
        super( score: 0);
        min = 1;
        max = 100;
    }

    @Override
    public int getGuess(){
        previousGuess = (max+min)/2;
        return previousGuess;
    }

    @Override
    public void responseToGuess(int response){
        if(response < 0){
            min = previousGuess;
        }
        else if(response > 0){
            max = previousGuess;
        }
    }

    @Override
    public String getName() { return "Smart Computer"; }
}
```

Interfaces

Et **interface** er en slags *kontrakt,* det fortæller **hvilke metoder** en klasse skal have — men **ikke hvordan** de skal implementeres.



Et interface kan kun indeholde konstanter (public static final), ikke almindelige attributter (felter)

En klasse kan implementere *flere* interfaces, men kun arve *én* klasse

Et inteface kan heller ikke instantieres

```
Item item1 = new Item();
Item item2 = new Food( name: "Apple", energy: 40, freshness: 100);
Item item3 = new Weapon( force: 50);
```

Eksempel på interface og implementering

```
3   public interface Item {  2 implementations
4
5       int use();  2 implementations
6   }
```

```
3   public class Food implements Item {
4
5       private int energy;
6       private String name;
7       private int freshness;
8
9       public Food(String name, int energy, int freshness){
10          this.name = name;
11          this.energy = energy;
12      }
13
14      public void reduceFreshness(){
15          freshness--;
16      }
17
18      public int use(){
19          return energy * freshness;
20      }
21  }
22
```

```
3   public class Weapon implements Item{
4
5       private int force;
6       private int damage;
7
8       public Weapon (int force){
9           this.force = force;
10      }
11
12      public void setDamage(){
13          this.damage = damage;
14      }
15
16      public int use(){
17          return force - damage;
18      }
19  }
```

Hvornår bruger vi hvad?

Abstrakt klasse

Behov for *fælles kode* og *struktur* → Giver *delvis færdig* baseklasse

Interface

Behov for *rolle / egenskab* → Definerer *hvad* en klasse kan gøre

Øvelse

Skriv interface og klasser med minimum metoden getPerimeter()

Circle perimeter:

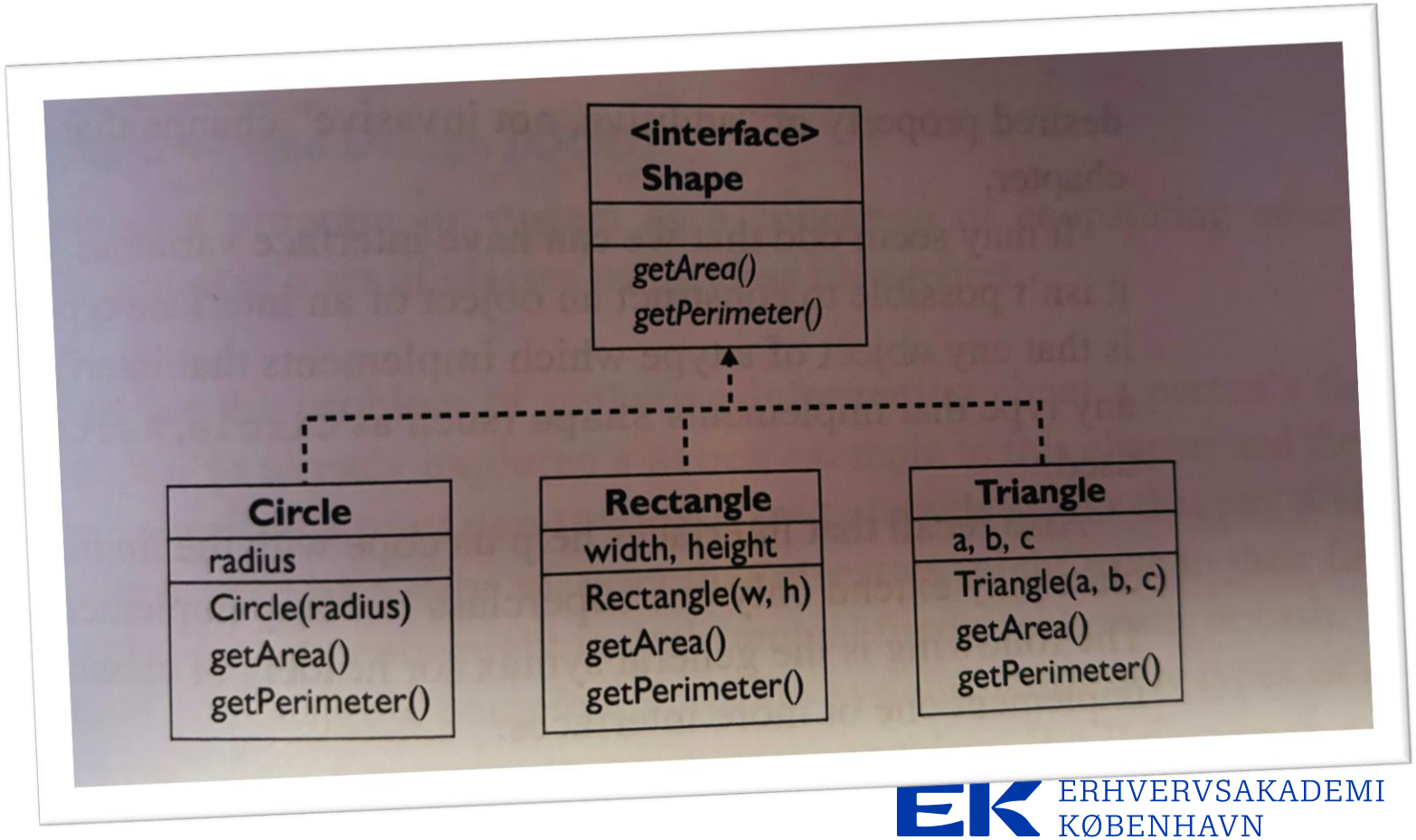
2 * PI * radius

Rectangle perimeter:

2 * (width + height)

Triangle perimeter:

a + b + c



Dynamic binding

- Vi kan behandle forskellige objekter ens, når de implementerer samme interface
- Vi kan kalde samme metode på en række objekter og få forskellig adfærd
- Det kalder vi **dynamic binding**
- Det er først på runtime at metodekaldet bindes til konkret kode, da det afhænger af det konkrete objekts type
- **Referencens type** bestemmer hvad der må kaldes og **objektets type** bestemmer hvilken kode der køres

```
ArrayList<Shape> shapes = new ArrayList();

double totalArea = 0;

for(Shape s: shapes){
    totalArea += s.getArea();
}
```

Referencens type kontra objektets type

```
Item apple = new Food( name: "Apple",  energy: 40,  freshness: 100);
int nutrition = apple.use();
apple.reduceFreshness();

Food bread = new Food( name: "Bread",  energy: 60,  freshness: 70);
int nutrition2 = bread.use();
bread.reduceFreshness();
```

Downcasting

Hvis vi har brug for at tilgå metoder fra subklassen, så kan vi downcaste til en variable af subklassens type

```
Item apple = new Food( name: "Apple",  energy: 40,  freshness: 100);
int nutrition = apple.use();
apple.reduceFreshness();

Food appleDownCastet = (Food) apple;
appleDownCastet.reduceFreshness();
```