Inheritance + Polymorphism

Hitesh Boinpally
Summer 2023

Agenda

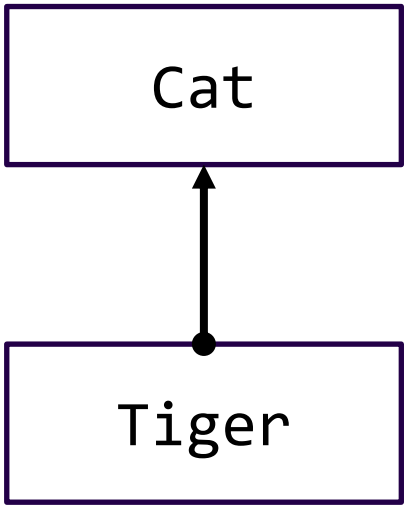
- Final Logistics
- Inheritance
- Different Errors
- Polymorphism

Inheritance

- **Inheritance:** Forming hierarchial relationships between classes
  - Allows for sharing / reusing of code between classes
  - **Superclass**: The class being extended
  - **Subclass:** The class that inherits behavior from superclass
    - Gains copy of every method

Inheritance

- **Inheritance:** Forming hierarchial relationships between classes
  - Allows for sharing / reusing of code between classes
  - **Superclass**: The class being extended
  - **Subclass:** The class that inherits behavior from superclass
    - Gains copy of every method
- Inheritance forms an "is-a" relationship
  - `Tiger extends Cat`
  - Means that `Tiger` "is-a" `Cat`



```
public class MusicPlayer {
    public void m1() {
        S.o.pln("MusicPlayer1");
    }
}
public class TapeDeck extends MusicPlayer {
    public void m3() {
        S.o.pln("TapeDeck3");
    }
}
```

```
public class IPod extends MusicPlayer {
    public void m2() {
        S.o.pln("IPod2");
        m1();
    }
}
public class IPhone extends IPod {
    public void m1() {
        S.o.pln("IPhone1");
        super.m1();
    }

    public void m3() {
        S.o.pln("IPhone3");
    }
}
```

	m1()	m2()	m3()
MusicPlayer			
TapeDeck			
IPod			
IPhone			

```
public class MusicPlayer {
    public void m1() {
        S.o.pln("MusicPlayer1");
    }
}
public class TapeDeck extends MusicPlayer {
    public void m3() {
        S.o.pln("TapeDeck3");
    }
}
```

```
public class IPod extends MusicPlayer {
    public void m2() {
        S.o.pln("IPod2");
        m1();
    }
}
public class IPhone extends IPod {
    public void m1() {
        S.o.pln("IPhone1");
        super.m1();
    }

    public void m3() {
        S.o.pln("IPhone3");
    }
}
```

	m1()	m2()	m3()
MusicPlayer	MP1	/	/
TapeDeck	MP1	/	TD3
IPod	MP1	IPod2 m1()	/
IPhone	IPhone1 MP1	IPod2 m1()	IPhone3

	m1()	m2()	m3()
MusicPlayer	MP1	/	/
TapeDeck	MP1	/	TD3
IPod	MP1	IPod2 m1()	/
IPhone	IPhone1 MP1	IPod2 m1()	IPhone3

```
MusicPlayer var1 = new TapeDeck();
MusicPlayer var2 = new IPod();
MusicPlayer var3 = new IPhone();
IPod var4 = new IPhone();
Object var5 = new IPod();
Object var6 = new MusicPlayer();


var1.m1();

var3.m1();

var4.m2();

var3.m2();

var5.m1();
```

	m1()	m2()	m3()
MusicPlayer	MP1	/	/
TapeDeck	MP1	/	TD3
IPod	MP1	IPod2 m1()	/
IPhone	IPhone1 MP1	IPod2 m1()	IPhone3

```
MusicPlayer var1 = new TapeDeck();
MusicPlayer var2 = new IPod();
MusicPlayer var3 = new IPhone();
IPod var4 = new IPhone();
Object var5 = new IPod();
Object var6 = new MusicPlayer();
```

var1.m1();
MusicPlayer1
var3.m1();
IPhone1 / MusicPlayer1
var4.m2();
IPod2 / IPhone1 /
MusicPlayer1
var3.m2();
Compiler Error (CE)
var5.m1();
Compiler Error (CE)

	m1()	m2()	m3()
MusicPlayer	MP1	/	/
TapeDeck	MP1	/	TD3
IPod	MP1	IPod2 m1()	/
IPhone	IPhone1 MP1	IPod2 m1()	IPhone3

```
MusicPlayer var1 = new TapeDeck();
MusicPlayer var2 = new IPod();
MusicPlayer var3 = new IPhone();
IPod var4 = new IPhone();
Object var5 = new IPod();
Object var6 = new MusicPlayer();

((TapeDeck) var1).m2();

((IPod) var3).m2();

((IPhone) var2).m1();

((TapeDeck) var3).m2();
```

	m1()	m2()	m3()
MusicPlayer	MP1	/	/
TapeDeck	MP1	/	TD3
IPod	MP1	IPod2 m1()	/
IPhone	IPhone1 MP1	IPod2 m1()	IPhone3

```
MusicPlayer var1 = new TapeDeck();
MusicPlayer var2 = new IPod();
MusicPlayer var3 = new IPhone();
IPod var4 = new IPhone();
Object var5 = new IPod();
Object var6 = new MusicPlayer();
```

((TapeDeck) var1).m2();
Compiler Error (CE)
((IPod) var3).m2();
IPod2 / IPhone1 / MusicPlayer1
((IPhone) var2).m1();
Runtime Error (RE)
((TapeDeck) var3).m2();
Compiler Error (CE)

The Rules

First we define a few things with a color code

```
DeclaredType name = new ObjectType();    //declare variable
name.method();                           //call method
((CastToType)name).method();             //cast object, then call a method
```

When we try to execute one of the latter two, we follow this progression: