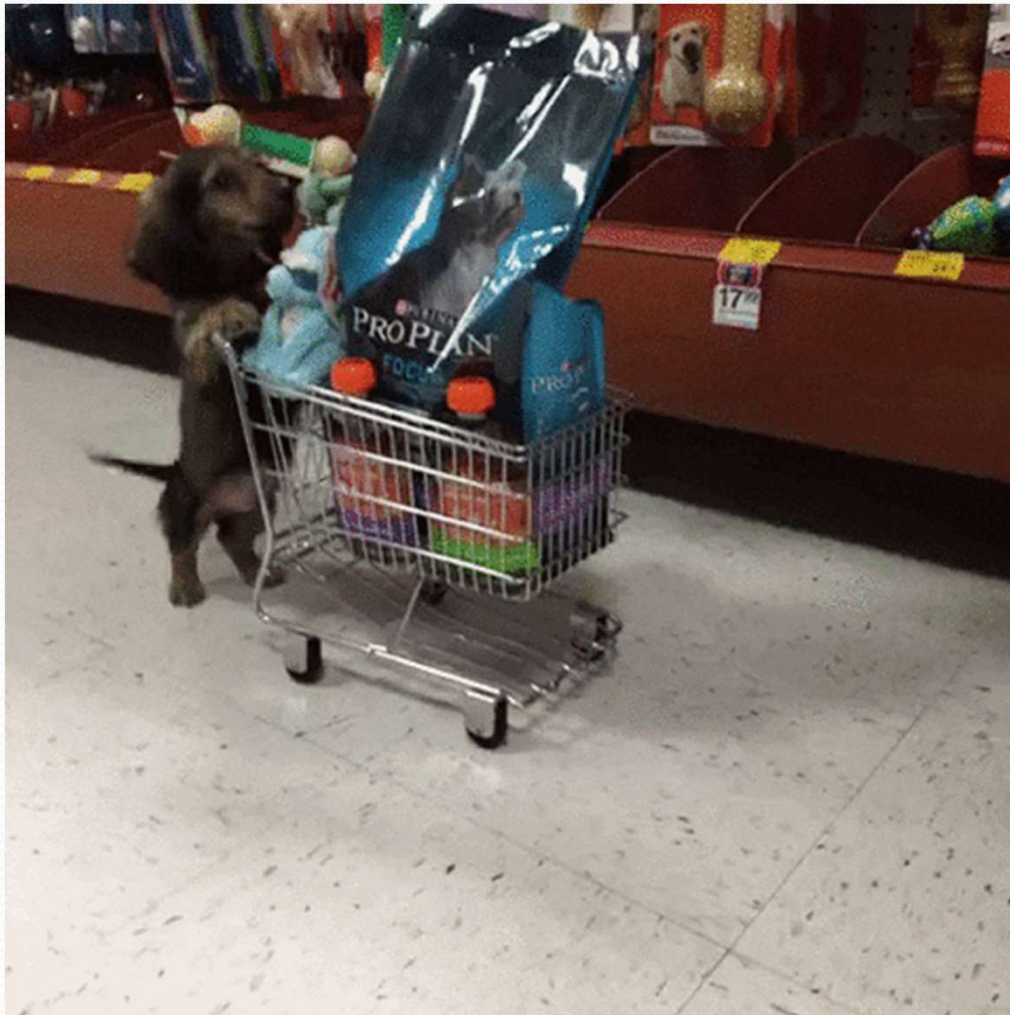




# Building Java Programs

Chapter 9  
Inheritance and Polymorphism

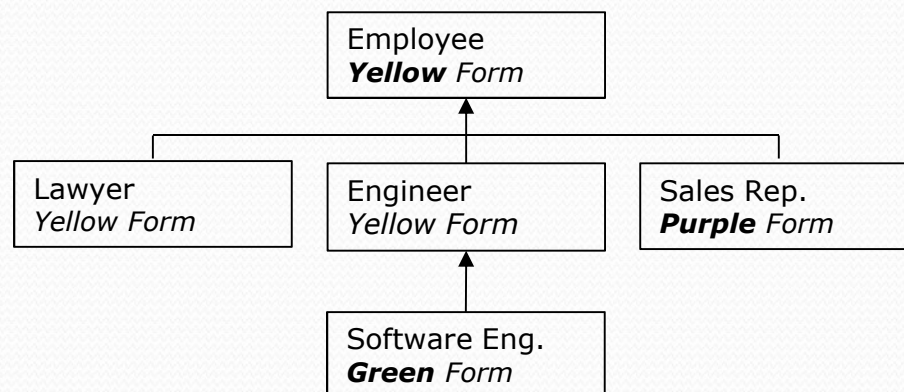
**reading: 9.1 - 9.2**





# Recall: Inheritance

- **inheritance**: Forming new classes based on existing ones.
  - a way to share/**reuse code** between two or more classes
  - **superclass**: Parent class being extended.
  - **subclass**: Child class that inherits behavior from superclass.
    - gets a copy of every field and method from superclass
- **is-a relationship**: Each object of the subclass also "is a(n)" object of the superclass and can be treated as one.



```
public class A {
    public void m1() {
        m2();
        S.o.pln("A1");
    }

    public void m2() {
        S.o.pln("A2");
    }
}

public class B extends A {
    public void m2() {
        S.o.pln("B2");
    }
}
```

## Poll Everywhere

```
B b = new B();
b.m1();
```

What is the output?

- A2 / A1
- B2 / A1
- Some kind of error
- I'm not sure



# Why cover this again?

- New Topics

- More practice with understanding polymorphism
- Investigating Java's type system
  - What happens when you using casting with objects?
  - What is and isn't possible for the compiler to check?

- Motivation: We've been hand-waving what it means to say

```
List<Integer> list = new ArrayList<Integer>();  
list.add(1);
```

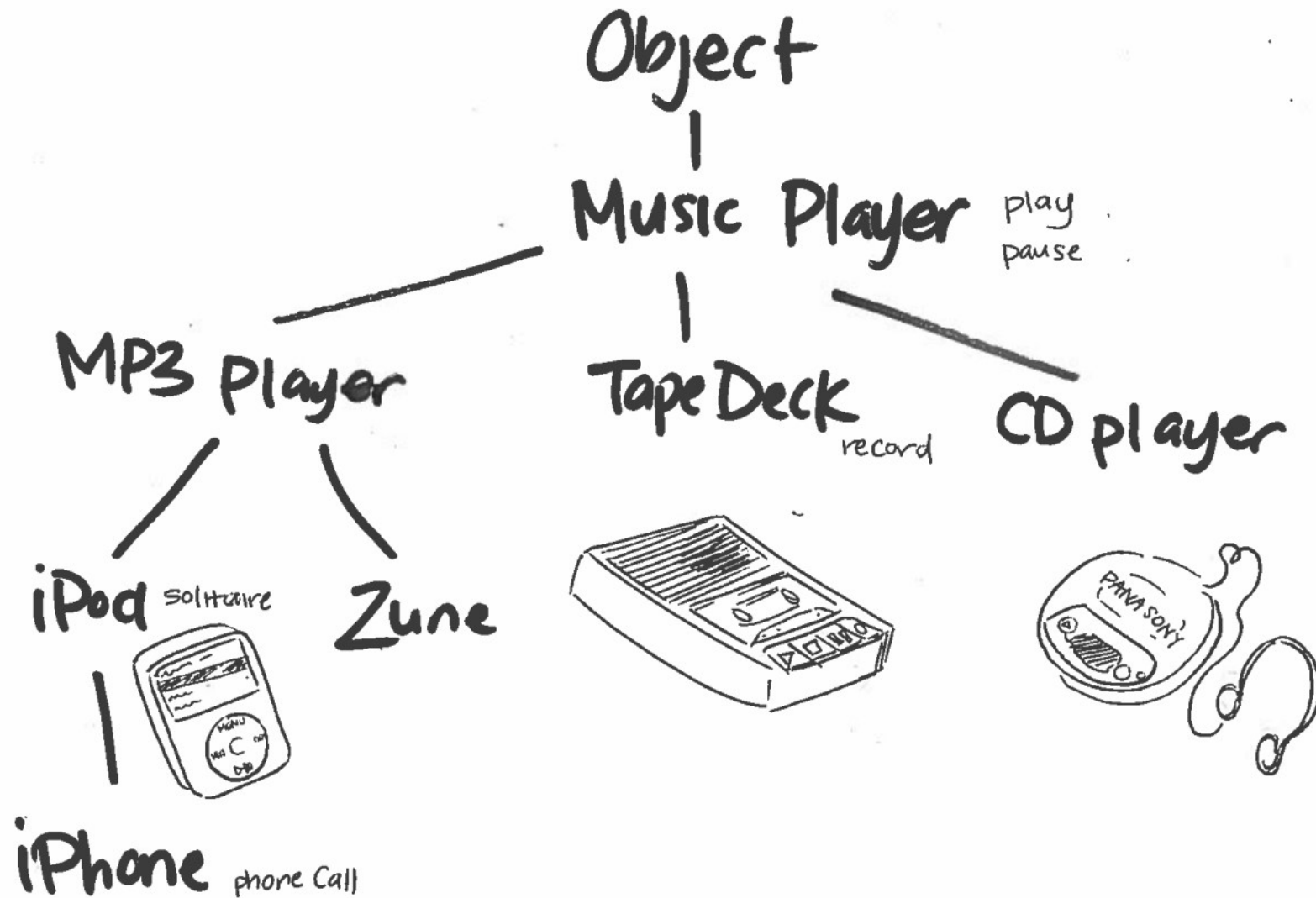
- Why allow different types on the left side vs. right side?

```
PromiseType variable = new ActualType();
```

- PromiseType can be a super-type that ActualType extends or an interface that ActualType implements

- Restricts usage of the instance of ActualType to only PromiseType methods. Why is this useful?

# Example: Music Players





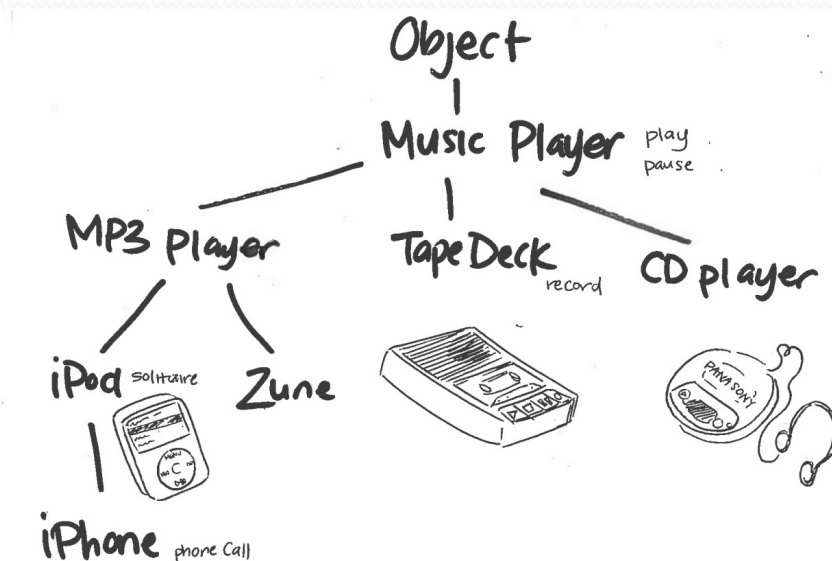
# Poll Everywhere

```
MediaPlayer p3 = new Zune();
```

```
((iPhone) p3).record();
```

What does this line do?

- Call record on Zune
- Call record on MediaPlayer
- Call record on iPhone
- Compiler Error
- Runtime Error



```

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	MPI	X	X
TapeDeck	MPI	X	TD3
IPod	MPI	IPod2 <u>m1()</u>	X
iPhone	IPhone1 <span style="border: 1px solid blue; padding: 2px;">MPI</span>	IPod2 <u>m1()</u>	IPhone3



	m1	m2	m3
MusicPlayer	MP1	/	/
TapeDeck	MP1	/	TD3
iPod	MP1	iPod2 <b>m1()</b>	/
iPhone	iPhone1 MP1	iPod2 <b>m1()</b>	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
MediaPlayer	MP1	/	/
TapeDeck	MP1	/	TD3
iPod	MP1	iPod2 <b>m1()</b>	/
iPhone	iPhone1 MP1	iPod2 <b>m1()</b>	iPhone3

```


MediaPlayer var1 = new TapeDeck();
MediaPlayer var2 = new iPod();
MediaPlayer var3 = new iPhone();
iPod var4 = new iPhone();
Object var5 = new iPod();
Object var6 = new MediaPlayer();


```

```

((TapeDeck) var1).m2();
Compiler Error (CE)

```

 ((iPod) var3).m2();  
**iPod2 / iPhone1 / MediaPlayer1**

 ((iPhone) var2).m1();  
**Runtime Error (RE)**

 ((TapeDeck) var3).m2();  
**Compiler Error (CE)**



# General Rule

```
PromiseType var = new ActualType();  
var.method()      or      ((CastType) var).method();
```

## Compile Time

```
if (involves casting) {  
    check if CastType has method, if not fail with CE  
} else {  
    check if PromiseType has method, if not fail with CE  
}
```

## RunTime (if compiles)

```
if (involves casting) {  
    check if ActualType can actually be cast to CastType,  
    if not fail with RE  
}  
call method on ActualType
```