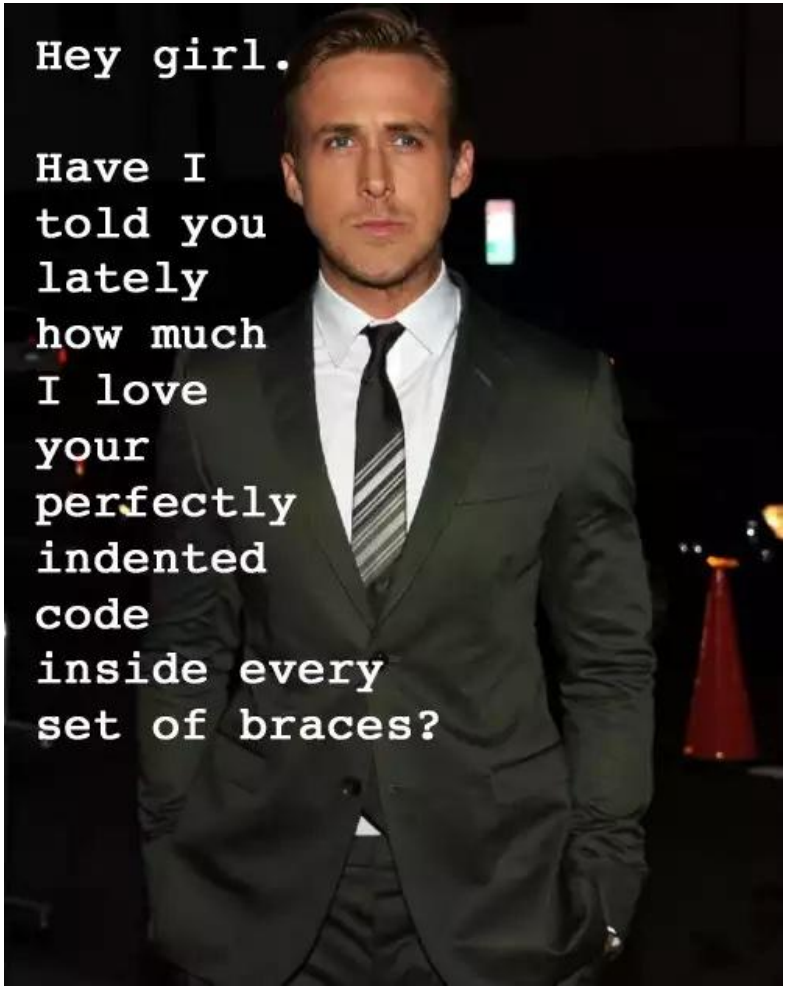
Inheritance

The Pinnacle Idea of JAVA


Hey girl.
Have I
told you
lately
how much
I love
your
perfectly
indented
code
inside every
set of braces?

Super Smash Brothers!!!

Which Smash Bros character are you?

http://www.playbuzz.com/hannahnoellegault10/which-super-smash-bros-character-are-you

http://goo.gl/7zE6X4

What are some sub-categories of characters?

What are some sub-categories of characters?

- Speed
- Power
- Technique
- Tricky
- Defense
- All-Around

Somebody Actually Wrote THIS:

The rest of the all-around types are: , , , , , , , , , and .

Next, Power types. These characters trade movement speed and multi-hitting and fast attacks for more weight and power.

For example, .
He has massive attack, but little movement speed (but much better in SSB4 then Brawl and Melee) or attack speed.
Generally, these characters are big, so they have good range too.

The rest are: , , , , , and .

Now, there are speed types. They are generally the nothing like power types. They overwhelm with fast movement and fast attacks with multi-hitters, generally with dashing specials and good recovery. They have little weight and power.

They are: , , , , , , , , , , , , , and .

Now is Technique. They have special things that help them.

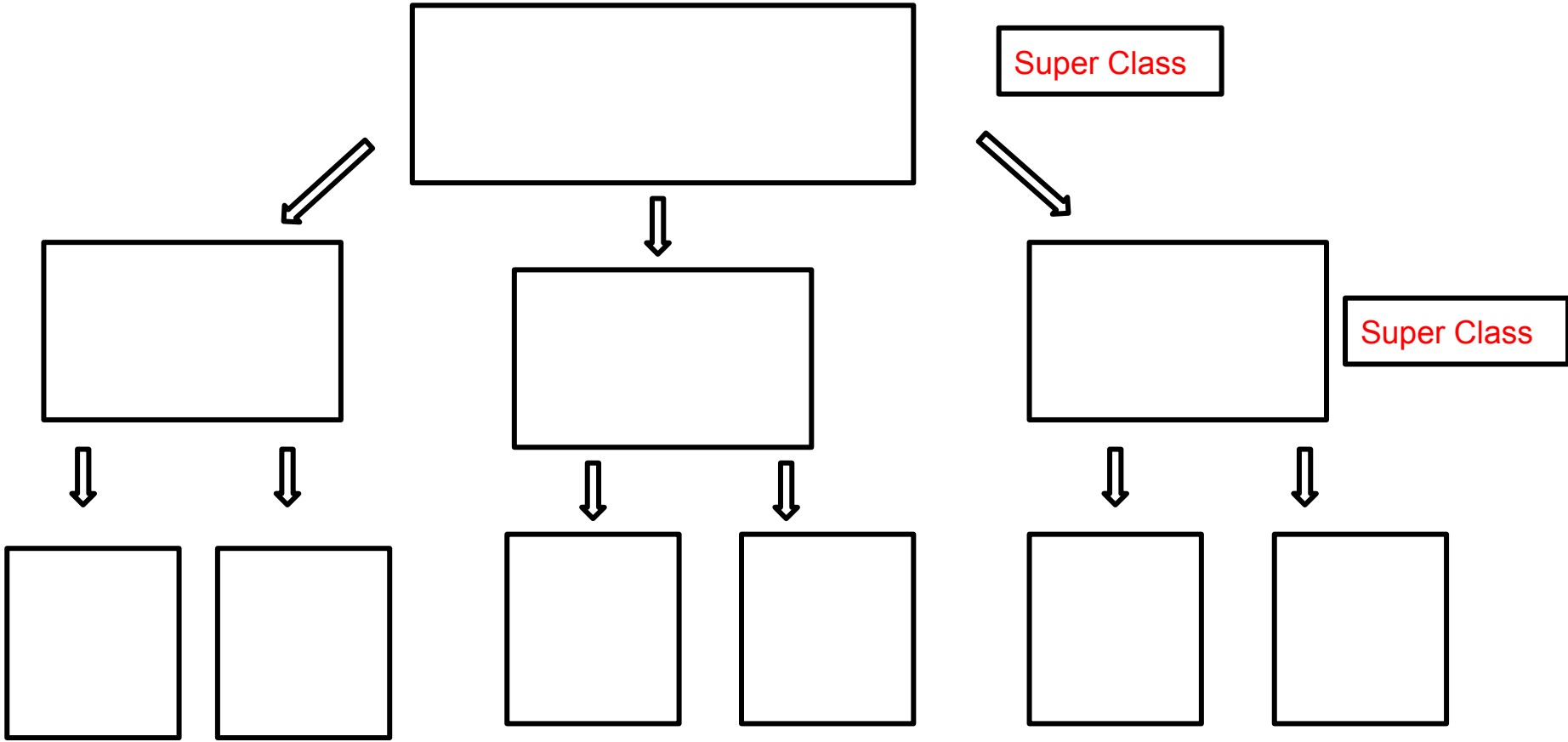
They are: , , , , , , , and .

Now there is Defence. They have a non-offence flow, with awesome recovery most of the time. Like Mario Power Tennis, there are vary few.

They are: , , , and .

Now is Tricky. It's hard to know what they're up to and you never know what's next. Or they make sure the match is played their way, even as CPUs. Ether they K.O. out of nowhere, you can't reach them no matter how hard you try, or they keep you close so you can't use projectiles easily.These should be called the s.

These characters are , , , , and


Super Class
Super Class

Smash Brothers Characters

Speed Chars

Technique Chars

Tricky Chars

Sonic

Pikachu

Kirby

Rosalina

Jigglypuff

Villager

Smash Brothers Characters

- getX(), getY() //find them
- moveX(), moveY() //move
- attack()

Speed Chars

- moveFast()

Technique Chars

- finesse()

Tricky Chars

- surprise()

Sonic
CrazyBall()

Pikachu
Dash()

Kirby
SwallowUp()

Rosalina
Aerial()

Jigglypuff
Sleep()

Villager
Umbrella()

How do we use this in Java? *extends*

```
public class SmashChar() {
    public int getX() { //code }
    public int getY() { }
    public void moveX(int x1)  { }
    public void moveY(int y1) { }
    public void attack () { }
}
```

```
public class SpeedChar()  extends SmashChar {
    public void speedUp()  {   }
}
```

```
public class Sonic()  extends SpeedChar {
    public void crazyBall()  {   }
}
```

Sonic is a subclass of SmashChar - he can use its methods

```
public class SmashChar() {
    public int getX() { //code }
    public int getY() { }
    public void moveX(int x1)  { }
    public void moveY(int y1) { }
    public void attack () { }
}
```

```
public class SpeedChar()  extends SmashChar
{
   public void speedUp()  {   }
}
```

```
public class Sonic()  extends SpeedChar
 {
   public void SpeedBall()  {   }
}
```

```
Sonic hedgeHog = new Sonic();
hedgeHog.SpeedBall();
hedgeHog.SpeedUp();
hedgeHog.getX();
hedgeHog.getY();
hedgeHog.moveX(3);
hedgeHog.moveY(7);
hedgeHog.attack();
```

We can also extend instance variables.

```
public class SmashChar() {
//This is the super class
  private String name;
  private int health;
  public SmashChar(String n, int h){
       name = n;
       health = h;
   }
}
```

```
public class Sonic()  extends SmashChar {
    private String color;
    public Sonic(String c, String n, int h) {
      color = c;
      super (n,h);
     }

}
```

What will be printed?

```
public class SmashChar() {
//This is the super class
  private String name;
  private int health;
  public SmashChar(String n, int h){
      name = n;
      health = h;
   }
  //gets and sets
}
```

```
Sonic hedgeHog = new Sonic("blue", "Sonny",9);
out.print (hedgeHog.getColor());
out.print(hedgeHog.getName());
out.print(hedgeHog.getHealth());
```

```
public class Sonic()  extends SmashChar {
    private String color;
    public Sonic(String c, String n, int h) {
      color = c;
      super (n,h);
     }
}
```

AP QUESTION

```
public class SmashChar() {
//This is the super class
  private String name;
  private int health;
  public SmashChar(String name){
      name = n;
      health = 100;
   }
  //gets and sets
}
```

```
JigglyPuff pinky = new JigglyPuff();
out.print(pinky.getName());
out.print(pinky.getHealth());
```

```
public class JigglyPuff()  extends SmashChar {
    public JigglyPuff( ) {
      super (“Jiggle”));
     }
}
```

Side Topic - Multiple Constructors

```
public class SmashChar() {
 private String name;
 private int health;
 public SmashChar( ){
     name = “SuperSmash!”;
     health = 10;
  }
 public SmashChar(String n, int h){
     name = n;
     health = h;
  }
}
```

```
public class JigglyPuff()  extends SmashChar {
   public JigglyPuff( ) {
     super () ;
    }
    public JigglyPuff(String n, int h ) {
     super (n, h) ;
    }
}
```

```
JigglyPuff pinky = new JigglyPuff();
out.print(pinky.getName());
out.print(pinky.getHealth());

JigglyPuff cute = new JigglyPuff(“JP”, 9);
out.print(cute.getName());
out.print(cute.getHealth());
```

Weird Topic - Needing to Cast

```
public class SmashChar() {
 private String name;
 private int health;
 public SmashChar( ){
     name = “SuperSmash!”;
     health = 10;
  }
 public SmashChar(String n, int h){
     name = n;
     health = h;
  }
}
```

```
public class JigglyPuff()  extends SmashChar {
   public JigglyPuff( ) {
     super () ;
    }
    public String giggle() {
       return “Giggle!”;
    }
}
```

```
SmashChar pinky = new JigglyPuff();
out.print(pinky.getName()); OK
out.print(pinky.getHealth()); OK
out.print(pinky.giggle()); NOT OK!!!!

out.print((JigglyPuff)pinky.giggle());
CASTING MAKES IT OK
```

Last Concept: Method Override

When you extend a class, you inherit all methods and instance variables.

You can override the original methods by implementing one with the same signature.

A signature is the method header like:

```
public int moveX(int change)
```

Example: Sonic's Speed

```
public class SpeedChar()  extends SmashChar {
    public void speedUp()  {
          speed += 20;
       }
}
```

```
public class Sonic()  extends SpeedChar {
    public void speedUp()  {
          speed += 30;
          speedBall(); }
}
```

Sonic's speedUp() overrides the standard speedUp() for SpeedChars.

He gets 30 points instead of 20 AND calls his special SpeedBall() method.

Example: Sonic's Speed

```
public class SpeedChar()  extends SmashChar {
    public void speedUp()  {
          speed += 20;
       }
  public String cheer() {
    return "Yay Smash!"
}
```

```
public class Sonic()  extends SpeedChar {
    public void speedUp()  {
          speed += 30;
          speedBall(); }
}
```

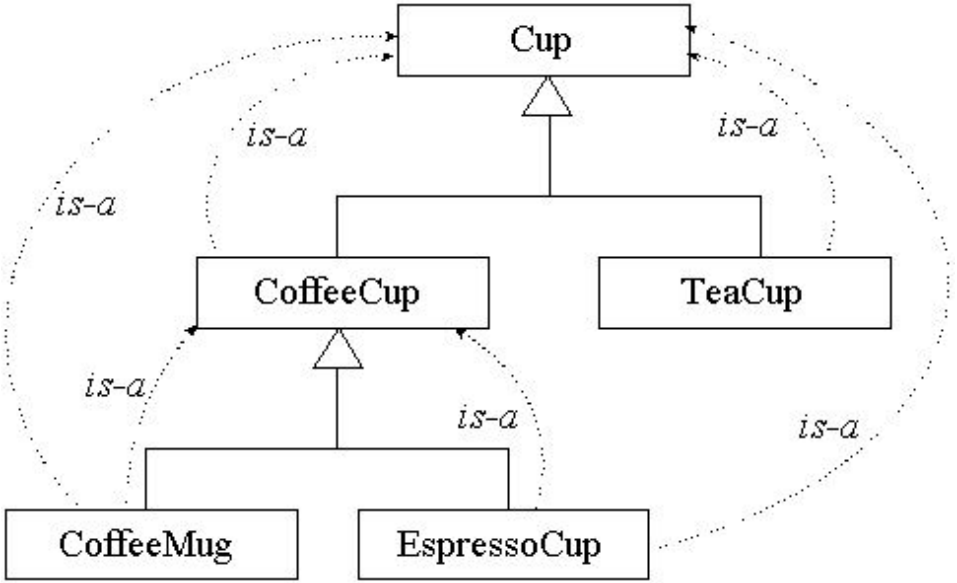
```
SmashChar hedgeHog = new
Sonic();
hedgeHog.speedUp();
  //this will call the "lowest"
  //version of speedUp() in Sonic
  //and stop there
  //Sonic's speedUP() overrides
  //that of SpeedChar

hedgeHog.cheer();
  //this will look in Sonic first
  //but he has no cheer()
  //so it will look "upward"
  //to find the first cheer()
```

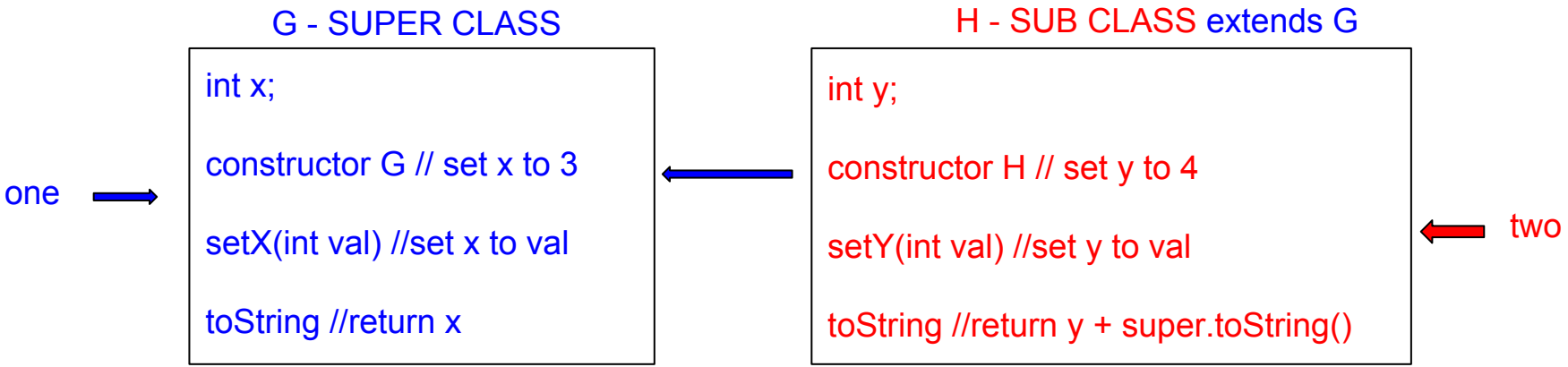
More Inheritance!

Inheritance diagram: “is a”

- CoffeeCup is a Cup
  - Cup is CoffeeCup’s “Parent” class
  - CoffeeCup is Cup’s “Child” class
- TeaCup and Coffee Cup are “siblings”
- Name another parent?
- Name another child?
- Name other sblings?



Drawings to explain Worksheet 2, #1




```
G one = new G();
out.println(one);
one.setX(5);
out.println(one);
```

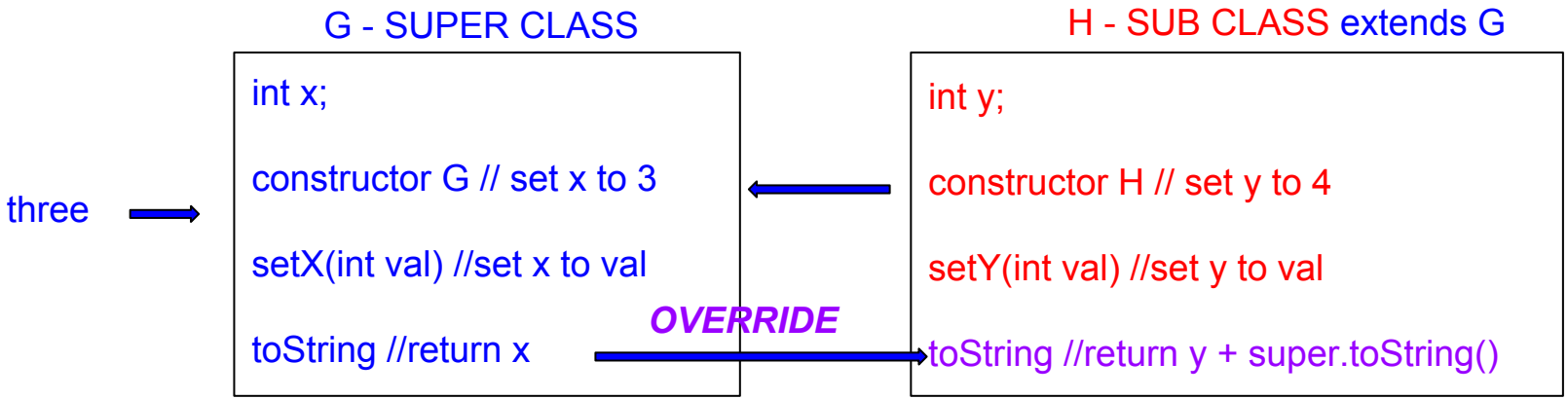
```
3

5
```

```
H two = new H();
two.setX(-2)
two.setY(11);
out.println(two);
```

```
11  -2
```

Drawings to explain Worksheet 2, #2

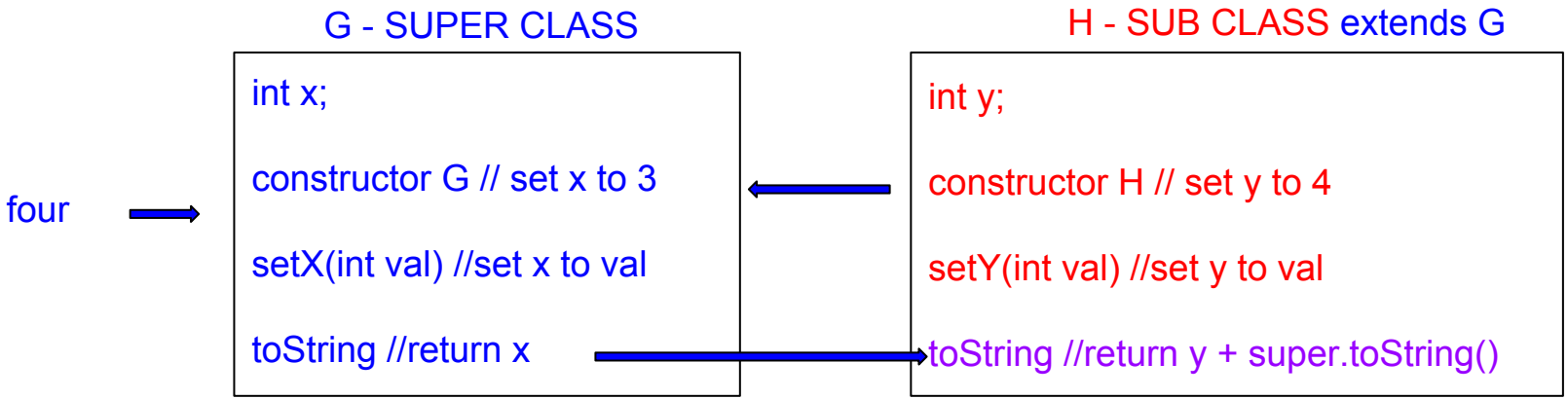



```
G three = new H();
out.println(three);
three.setX(8);
three.setY(21);
out.println(three);
```

The answer is
“ERROR”

Which line causes it?

Drawings to explain Worksheet 2, #3

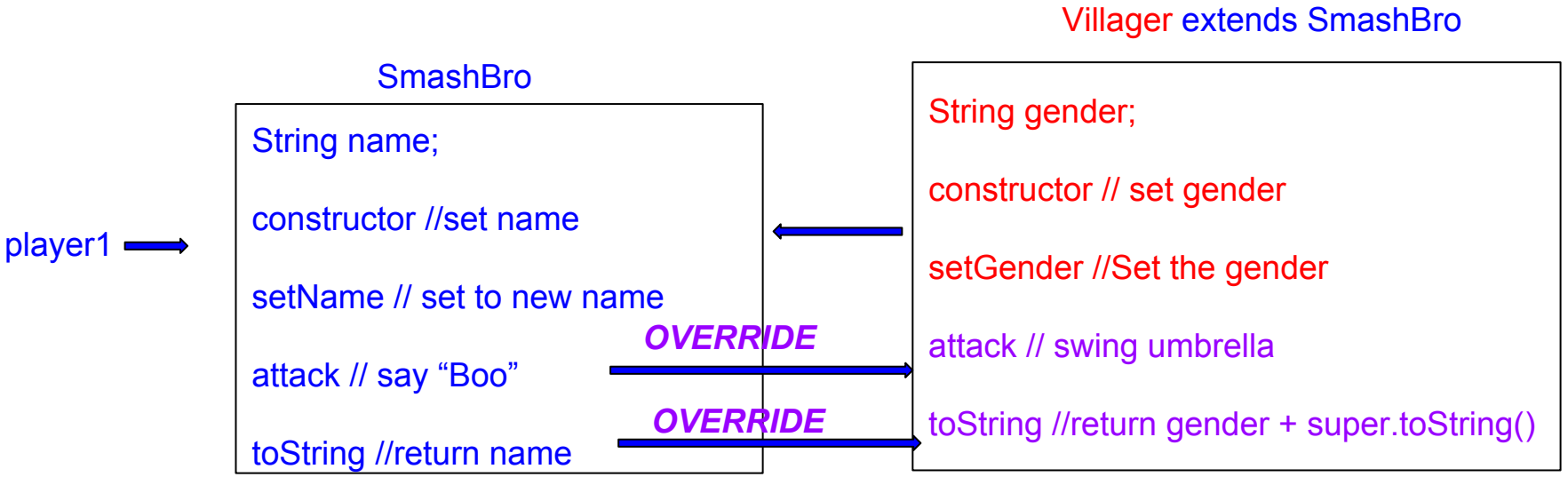



```
G four = new H();
four.setX(11);
out.println(four);
four.setX(6);
((H)four).setY(14);
out.println(four);
```

```
4  11

14  6
```

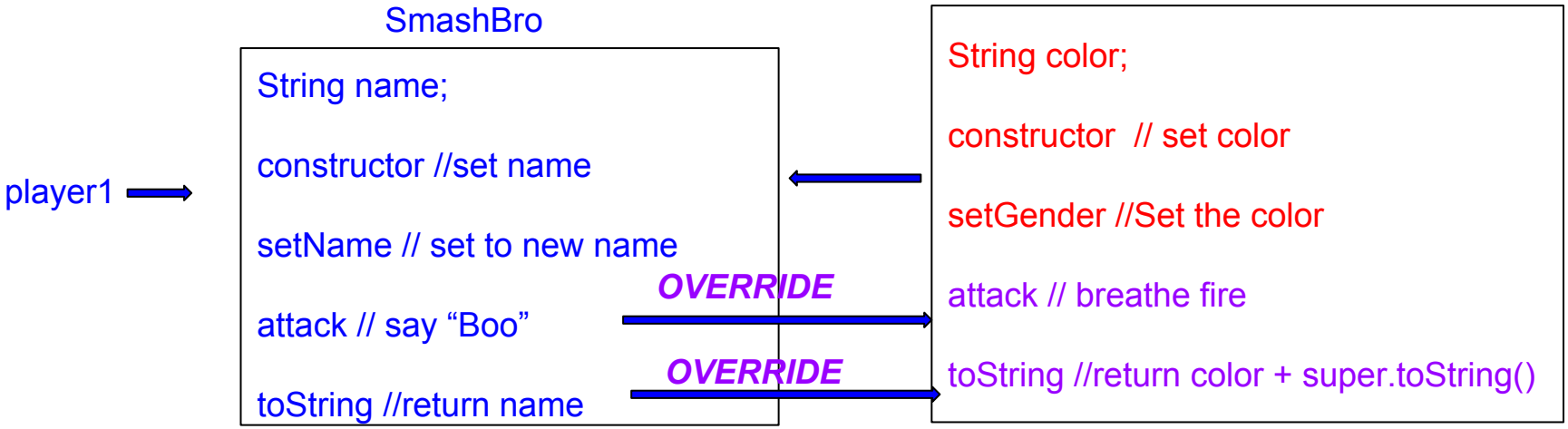
Why didn't we just start as an H in the first place?




```
SmashBro player1 = new Villager();
player1.setName("Beyonce");
((Villager)player1).setGender("Female");
player1.attack() //What will this do?
out.print(player1);  //What will this print?
```

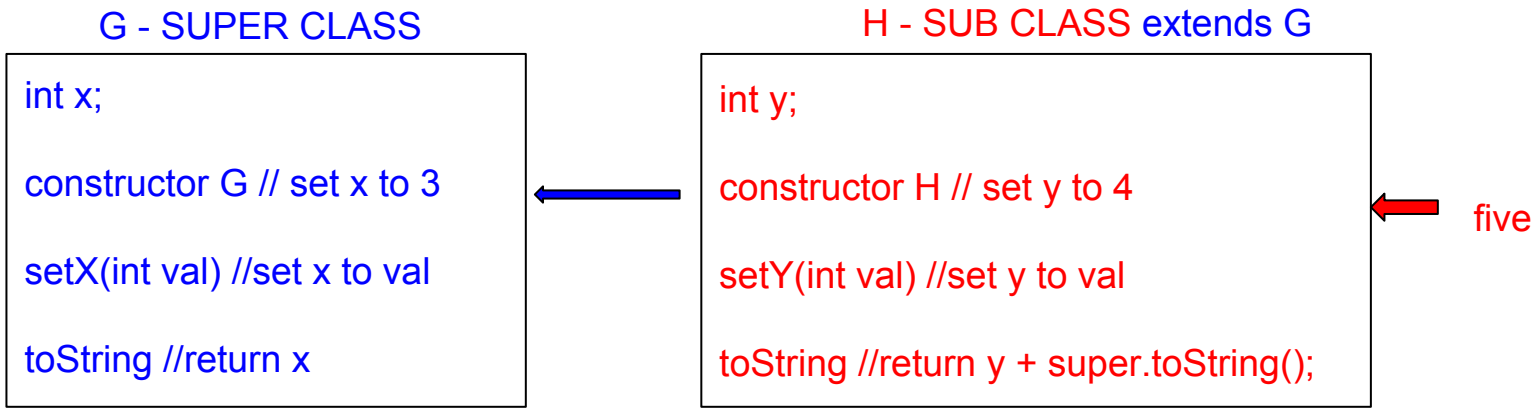
So the player can change personas!

Charizard extends SmashBro




```
SmashBro player1 = new Charizard();
player1.setName(“Beyonce”);
((Charizard)player1).setColor(“Red”);
player1.attack() //What will this do?
out.print(player1);  //What will this print?
```

Drawings to explain Worksheet 2, #4



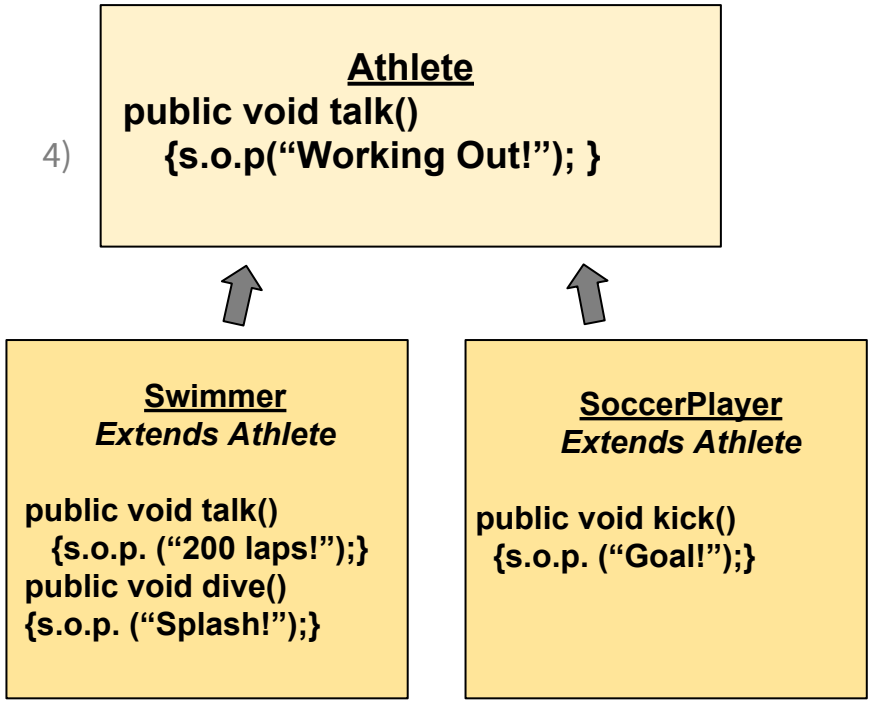

```
H five = new H();
five.setY(7);
out.println(five);
five.setX(16);
five.setY(9);
out.println(five);
```

```
7   3


9   16
```

Warmup: SUPER TRICK AP QUESTION: Override

4)



For each method call, write the output or "error"

1) Athlete ben = new Athlete();
ben.talk();
ben.kick();

2) SoccerPlayer sam = new SoccerPlayer();
sam.talk();
sam.kick();

3) Athlete poppy = new SoccerPlayer();
poppy.talk();
poppy.kick();

Warmup: SUPER TRICK AP QUESTION: Override

4)

Athlete
```
public void talk()
   {s.o.p("Working Out!"); }
```

Swimmer
Extends Athlete
```
public void talk()
   {s.o.p. ("200 laps!");}
public void dive()
{s.o.p. ("Splash!");}
```

SoccerPlayer
Extends Athlete
```
public void kick()
  {s.o.p. ("Goal!");}
```

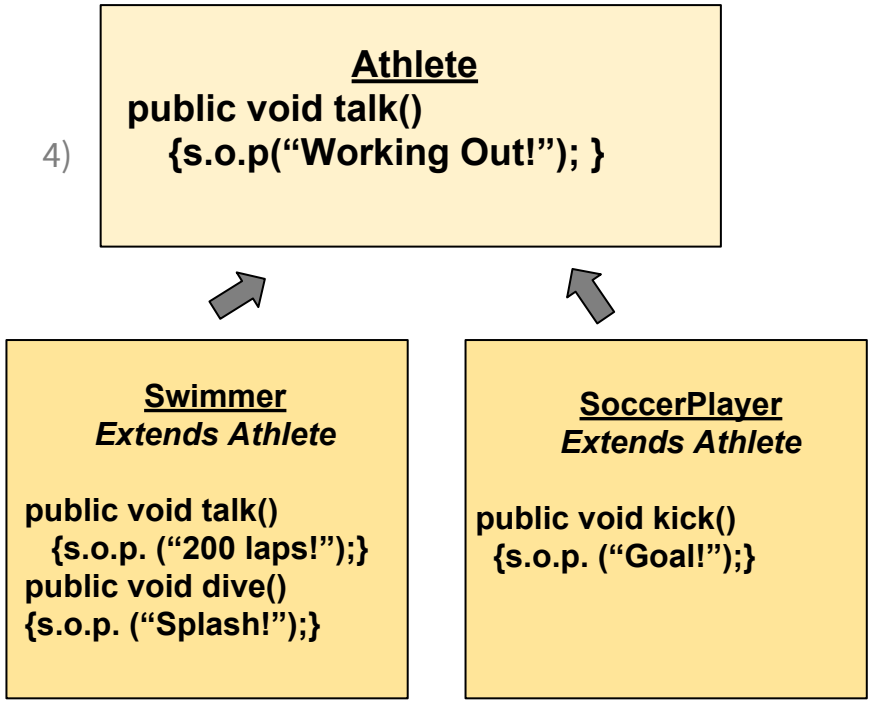
For each method call, write the output or "error"

1) `Athlete ben = new Athlete();`
   `ben.talk();` would say Working Out! except...
   `ben.kick();` ERROR! Athletes can't kick

2) `SoccerPlayer sam = new SoccerPlayer();`
   `sam.talk();` Working Out!
   `sam.kick();` Goal!

3) `Athlete poppy = new SoccerPlayer();`
   `poppy.talk();` Working Out!
   `poppy.kick();` ERROR! There's no override

SUPER TRICK AP QUESTION: Override



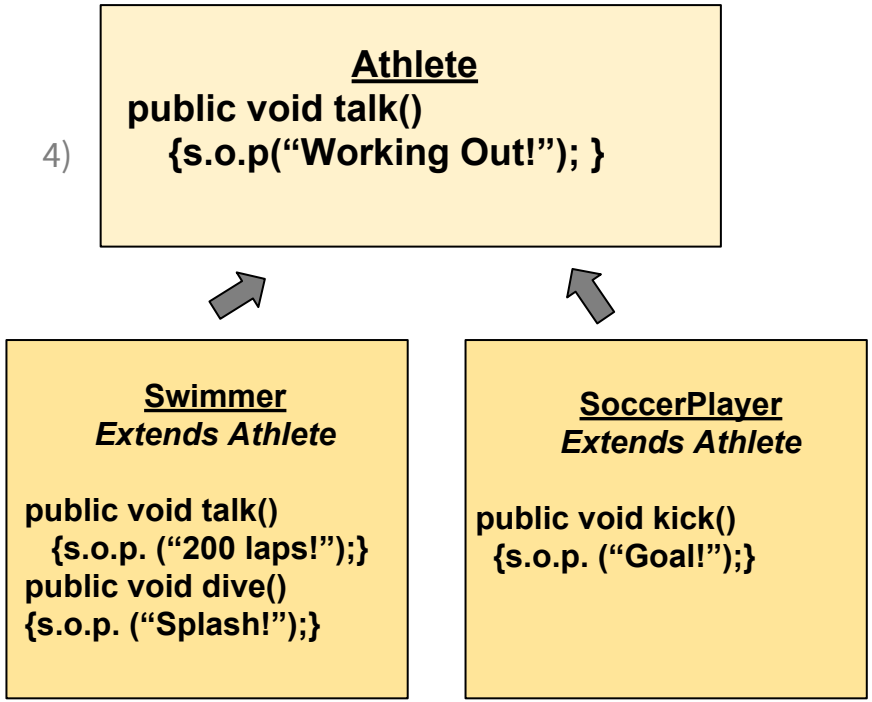
For each method call, write the output or "error"

```
4) Swimmer   emma  = new Swimmer();
        emma.talk();
        emma.dive();

 5)   Athlete  zuzu  = new Swimmer ();
        zuzu.dive();

6)     Athlete   lola  = new Swimmer();
        lola.talk();
```

SUPER TRICK AP QUESTION: Override



For each method call, write the output or "error"

4) Swimmer emma = new Swimmer();
emma.talk(); 200 laps!
emma.dive(); Splash!

5) Athlete zuzu = new Swimmer ();
zuzu.dive(); Error! Dive has no override

6) Athlete lola = new Swimmer();
lola.talk(); 200 laps! //override

Methods: Override vs Overload

Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name, Same parameter

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name, Different Parameter

Override Example

What is the output?

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}

public class OverridingTest{
    public static void main(String [] args){
        Dog dog = new Hound();
        dog.bark();
    }
}
```

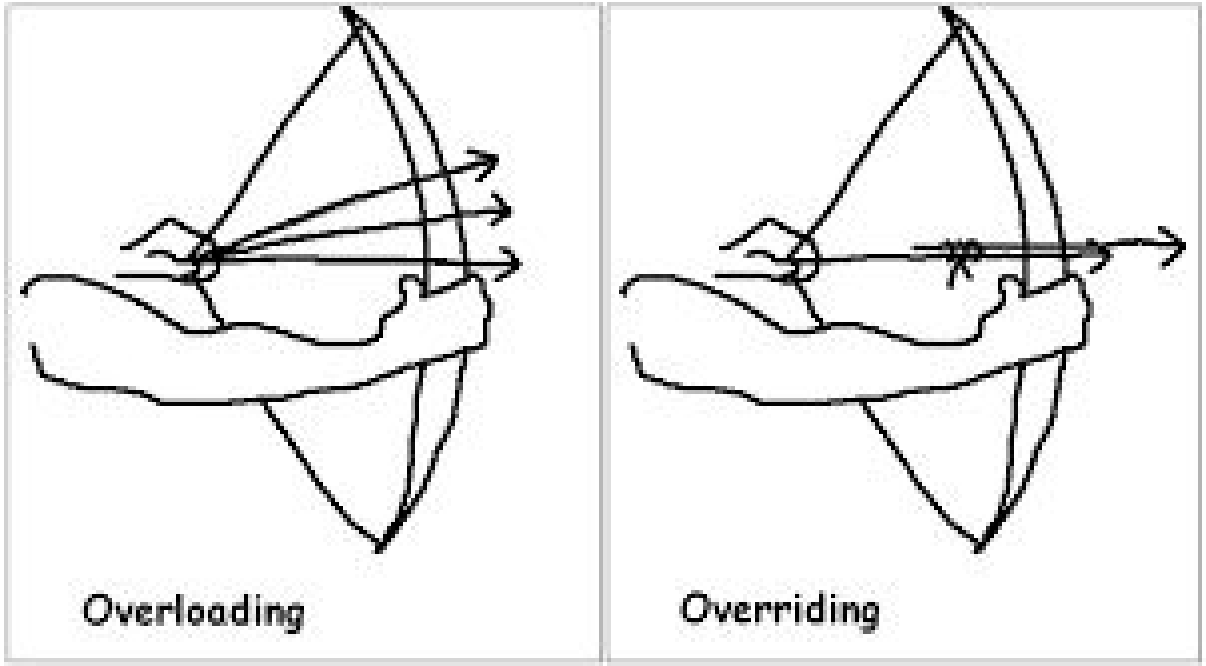
Overload example. What is output?

1) bark() 2) bark(5)

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
                System.out.println("woof ");
    }
}
```

Great Visual!



Warm-Up: Interface vs Inheritance?

Interface and Inheritance are both important parts of Polymorphism - they allow different types to use the same method calls to standardize behavior.

When might we choose one or the other?

Which one is interface: contract with abstract methods?
Which is inheritance: super class w/ concrete methods?

You're working on an animal video game with your best buddies. You're each in charge of one animal. You agree that all of the animals should have these methods: eat(), sleep(), talk()

You go to Cal and you're being forced to collaborate with Stanford students on an animal game. Stanford is making a bird, and you are making a bear. They should both eat(), sleep(), and talk(). You don't want the Stanford students to see your awesome code and steal it.

Warm-Up: Coding Bat

String-3 > withoutString

prev | next | chance

Given two strings, **base** and **remove**, return a version of the base string where all instances of the remove string have been removed (not case sensitive). You may assume that the remove string is length 1 or more. Remove only non-overlapping instances, so with "xxx" removing "xx" leaves "x".

withoutString("Hello there", "llo") → "He there"
withoutString("Hello there", "e") → "Hllo thr"
withoutString("Hello there", "x") → "Hello there"

Buddies use INHERITANCE - we can agree on and share the same super code.

Class Animals
eat() {o.p.(”Yum!”) }
sleep() {o.p.(”zzzzz”) }
talk() {o.p.(”Noise!”) }

class Tiger
talk() {o.p.(“roar!”);}

class Koala
talk() {o.p.(“yawn!”);}

*Tiger and Koala **extend** Animal. They can use the super classmethods to eat and sleep. We are **overriding** talk() to be more specific.*

Cal v Stanford uses an interface - we can agree on methods but don’t need to share code. This is ENCAPSULATION.

Interface Animals
eat()
sleep()
talk()

class Cardinal
eat() {o.p.(“seed!”);}
sleep() {o.p.(“zz”);}
talk() {o.p.(“peep”);}

class Bear
eat() {o.p.(“meat!”);}
sleep() {o.p.(“zz”);}
talk() {o.p.(“grrrr”);}

*Cardinal and Bear **implement** Animal. They need their OWN concrete methods to match the abstract method signatures.*

Methods: Override vs Overload

Override:
Used in inheritance.
A sub class can use the same *method signature* as a parent class method.

Overload:
A class can use the same *method name* with different parameters. lists example:
remove(Object obj);
remove(int i);