Victoria University of Wellington, New Zealand
Te Whare Wananga o te Upoko o te Ika a Maui Aotearoa

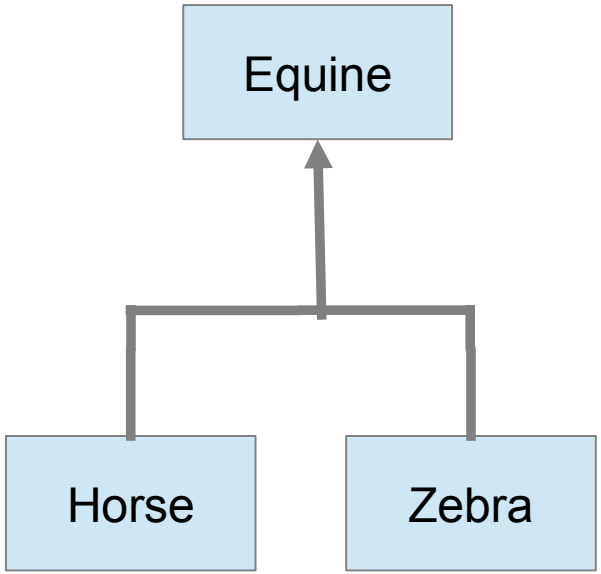
SWEN221: Software Development

7: Polymorphism I

David J. Pearce & Marco Servetto
Computer Science, Victoria University

Example class hierarchy

```
abstract class Equine {
  private Point position;
  public void run() {position.x+=1;}
  abstract void draw();
}
class Horse extends Equine{
  public void draw(){…}
}
class Zebra extends Equine{
  public void draw(){…}
}
```



Superclass: Equine

Subclass: Horse, Zebra

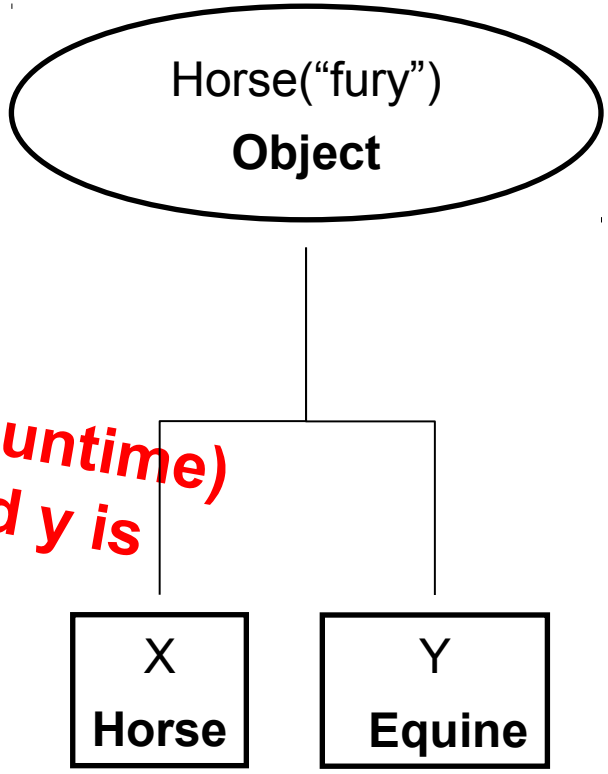
Mental Model of Typing

```
abstract class Equine {...}
class Horse extends Equine{...}
class Zebra extends Equine{...}

Horse x=new Horse();
Equine y=x;
```

Static (or Declared) type of y is "Equine"

Dynamic (or Runtime) type of x and y is "Horse"



- Static Type:
  - *Declared* type of a variable
- Dynamic Type:
  - Type of object *referred* to by variable
    - potentially different in any moment

Inheritance / Subtype polymorphism

Main concept for polymorphism in object oriented.

Horse and Zebra instances should be usable everywhere an instance of an Equine type is required.

A ColoredPoint instance should be usable everywhere an instance of Point Type is required.

Inheritance / Subtype polymorphism

More in general

Sublcasses instances should be usable everywhere a Supertype is required.

SHOULD means that YOU have to ensure this while writing the program.

- **Do not abuse instanceof and casts**
- **Understand and keep in mind the meaning of overriding a method**

What's wrong with this?

```
class ManageEquine {
  static float weightOfEquine(Equine q) {
    if (q instanceof Horse) return 250f;

    if (q instanceof Zebra) return 200f;

    if (q instanceof Pegasus) return 310f;

    throw new ThisIsNotHappeningError();
     }
}
```

What's wrong with this?

Someone can always add yet another subclass of Equine.

Ok, but, now, it works, right?

So what?

If in the future I add another subclass of Equine, I will add another "if" to that method.

What's wrong with this?

Will you remember to do so?

Will you be still in charge to maintain such code?

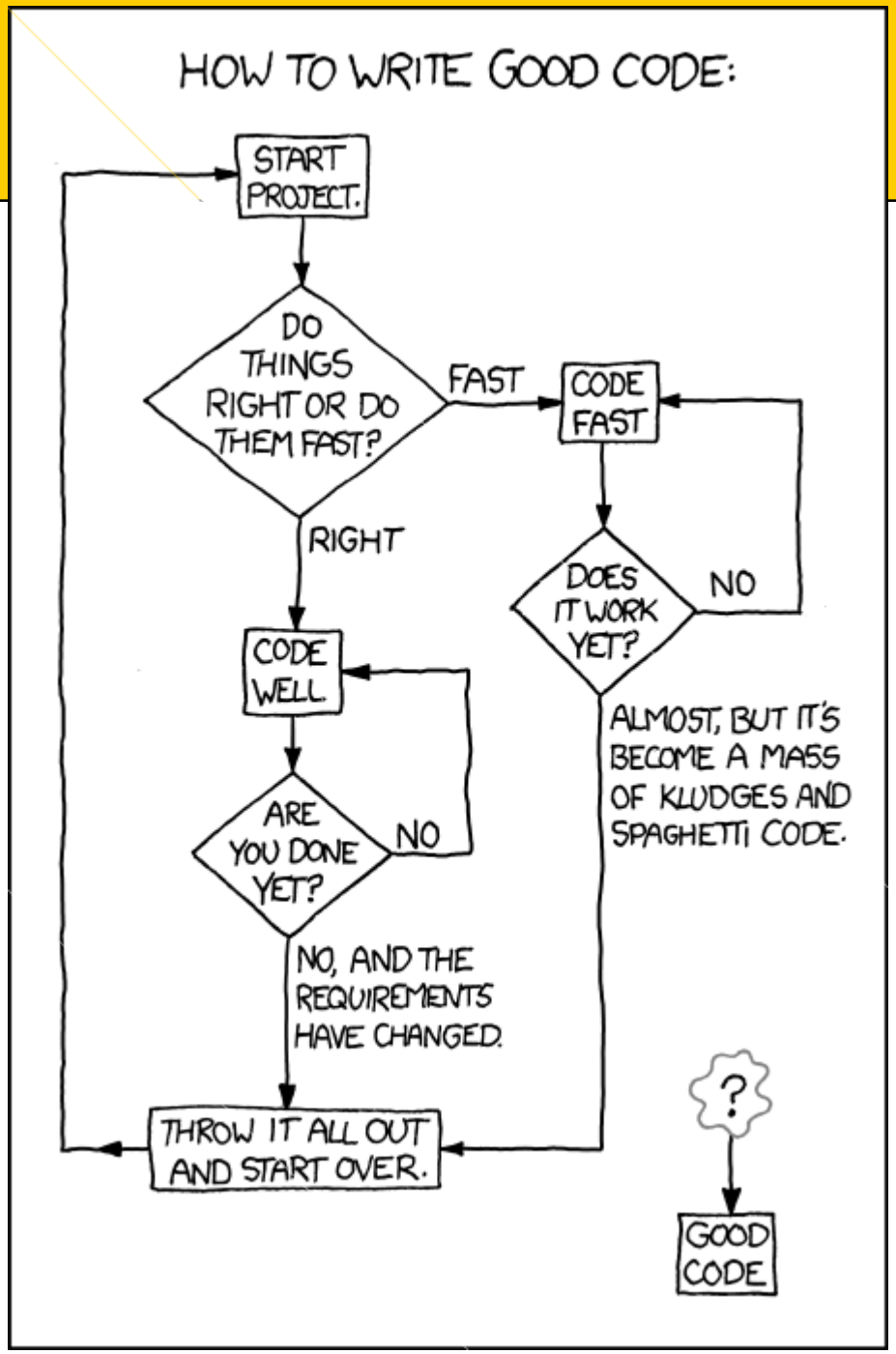
And what if your code is a library and is the library user that adds the Equine subclass?

What's Really wrong with this?

It's the mental setting that is profoundly wrong:

Inexpert programmers usually see the code just as a instrument to solve a concrete, contingent, problem.

http://xkcd.com/844/



The good mental setting

Write a (little?) set of libraries solving the category of problems in the problem domain.

The "real" program is just a little layer of code invoking (also) your libraries.

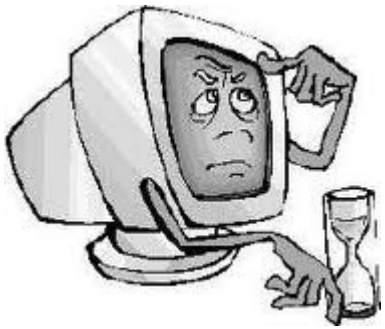
Good solution - Use overriding

```
abstract class Equine {...
  abstract float weight();
  }
class Horse extends Equine{...
  float weight(){return 200f;}
  }
...
```

Good solution - Use overriding

```
Equine x=....

x.weight()
```



```
Equine x=new Horse()

x.weight()
```

```
class Horse{
   ...
   float weight(){
     return 200f;}
   ...
   }
```

```
Equine x=....

200f
```

Dynamic Dispatch

```
x.someMethod(y);
```

`x` is the receiver

`y` is the parameter

- Method dispatch:
  - Two phases – compile time and runtime
- Static checking phase (overloading resolution):
  - Based on static types of receiver and parameters
  - Can only call methods defined in static type of receiver
- Dynamic dispatch (overriding resolution):
  - Selection of method at runtime
  - Dynamic choice depends only on receiver

Dispatch 1: static checking

- Identify ***static*** type of receiver
- Find methods that could possibly apply
  - Correct name
  - Correct number of arguments
  - Argument types that could apply
    - By looking at ***Static types***
    - Take into account type conversions
  - Search super-types
- Choose the most specific method (overloading)

Dispatch 1: static checking

Method names and static types

Not return types

As result of this phase you have a

method descriptor

The method ***descriptor*** is chosen statically

Dispatch 1: static checking

What is a method descriptor?

As an example

m(Foo,Bar,int)

It is a methodName followed by the statically declared type of the parameters.

Descriptors are the “overloaded resolved” version of a method name.

If you have found **m(Foo,Bar,int)** as result of phase 1, it means that some class **define** a method **m** with **Foo Bar** and **int** as parameter types

Dispatch 2: dynamic dispatch

To determine which method is called at runtime:

a: Identify ***dynamic*** type of receiver

b: Check for the method implementation using the method *descriptor*

- Remember, this used static types of arguments

If the method descriptor is not there, then look in super type(s) and then its super type(s) until match

Since the descriptor is from phase 1, you will find one match.

Quiz

```
class Cat {
 String whatAmI() {
  return "I'm a Cat!";
 }
 void print() {
  System.out.println(whatAmI());
}}
class Kitten extends Cat {
 String whatAmI() {
  return "I'm a Kitten!";
}}
Cat gypsy = new Cat();
Cat spike = new Kitten();
gypsy.print();
spike.print();
```

A) "I'm a Kitten!"
   "I'm a kitten!"

B) "I'm a Cat!"
   "I'm a Kitten!"

C) "I'm a Cat!"
   "I'm a Cat!"

Quiz

```
class Cat {
 public void action(Cat c) {
   System.out.println(1);}
 public void action(Kitten c) {
   System.out.println(2);}}

class Kitten extends Cat {}

Cat gypsy = new Cat();
Cat spike = new Kitten();
Kitten teddy = new Kitten();
gypsy.action(spike);
spike.action(teddy);
teddy.action(teddy);
```

A)
1
2
2

B)
1
1
2

Quiz

```
class Cat {
 public void action(Cat c) {
   System.out.println(1);}}

class Kitten extends Cat {
 public void action (Kitten k) {
   System.out.println(2);}}

Cat gypsy = new Cat();
Cat spike = new Kitten();
Kitten teddy = new Kitten();
gypsy.action(teddy);
spike.action(teddy);
teddy.action(teddy);
```

A)
```
1
2
2
```

B)
```
1
1
2
```

Quiz

```
class Cat {
 public void action(Kitten c) {
   System.out.println(1);}}

class Kitten extends Cat {
 public void action(Cat k) {
   System.out.println(2);}}

Cat gypsy = new Cat();
Cat spike = new Kitten();
Kitten teddy = new Kitten();
gypsy.action(teddy);
spike.action(teddy);
teddy.action(teddy);
```

A)
```
1
1
2
```

B)
```
1
1
1
```

Quiz

```
class Cat {
  public void action(Cat c, Kitten k) {
    System.out.println(1);}}

class Kitten extends Cat {
  public void action(Kitten k, Cat c) {
    System.out.println(2);}}

Cat gypsy = new Cat();
Cat spike = new Kitten();
Kitten teddy = new Kitten();
gypsy.action(spike,teddy);
spike.action(teddy,spike);
teddy.action(teddy,teddy);
```

A) 1, 1, 2

B) **compilation error**

C) 1, 2, 2

Quiz

```
class Pow {
  public int pow(int base,int exp) {
    if(exp==0)return 1;
    return base*pow(base,exp-1);
}}

class PowLog extends Pow {
  public int pow(int base,int exp) {
    System.out.println("LogMessage");
    return super.pow(base,exp);}
  }

System.out.println(new PowLog().pow(4,3));
```

A)
```
LogMessage
64
```

B)
```
LogMessage
LogMessage
LogMessage
64
```