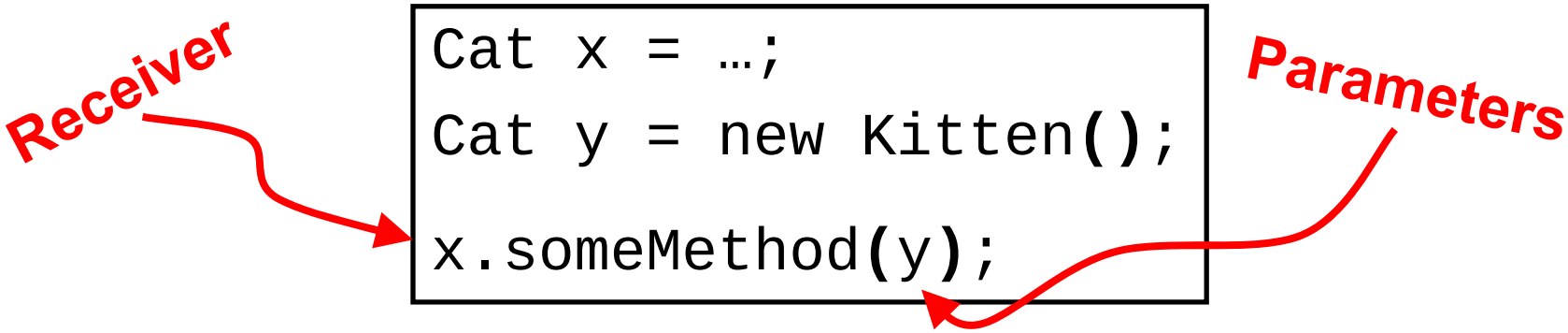
Victoria University
of Wellington, New Zealand
Te Whare Wananga o te
Upoko o te Ika a Maui
Aotearoa

SWEN221: Software Development

10: Polymorphism II

David J. Pearce & Nicholas Cameron & James Noble & Petra Malik
Computer Science, Victoria University

Dynamic Dispatch



- Method dispatch:
  - Two phases – compile time (static) and runtime (dynamic)
- Static checking phase:
  - Based on static types of receiver and parameters
  - Can only call methods defined in static type of receiver
- Dynamic dispatch:
  - 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…

Dispatch 1: static checking

- Choose the most specific method
  - The method *descriptor* is chosen statically
  - Like a method name but includes information about arguments
  - No overloading for descriptors

- Look at method names and static types
  - **Not** return types

Method Overloading

```
class C {
   void m(Cat c) {…}

   void m(Kitten k) {…}
}
```

- Here, `m(Kitten)` **overloads** `m(Cat)`
  - They have same name, but **different signature**
  - Overloading contrasts with **overriding**
  - i.e. overloaded methods *cannot override each other*

More Dispatch Examples

```
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!”

Dispatch 2: dynamic dispatch

To determine which method called at runtime:

1. Identify ***dynamic*** type of receiver
2. Check for method using descriptor
   - Remember, this used static types of arguments
3. If not, look in super type(s) and then its super type(s) until match

Quiz

```
class Cat {
 public void isClawedBy(Cat c) {
  System.out.println("Clawed by a Cat!");
}
 public void isClawedBy(Kitten c) {
  System.out.println("Clawed by a Kitten!");
}}
class Kitten extends Cat {}
Cat gypsy = new Cat();
Cat spike = new Kitten();
Kitten teddy = new Kitten();
gypsy.isClawedBy(spike);
spike.isClawedBy(teddy);
teddy.isClawedBy(teddy);
```

A)
```
"Clawed by a Cat!"
"Clawed by a Kitten!"
"Clawed by a Kitten!"
```

B)
```
"Clawed by a Cat!"
"Clawed by a Cat!"
"Clawed by a Kitten!"
```

Quiz

```
class Cat {
 public void isClawedBy(Cat c) {
  System.out.println("Clawed by a Cat!");
}}
class Kitten extends Cat {
 public void isClawedBy(Kitten k) {
  System.out.println("Clawed by a Kitten!");
}}
Cat gypsy = new Cat();
Cat spike = new Kitten();
Kitten teddy = new Kitten();
gypsy.isClawedBy(teddy);
spike.isClawedBy(teddy);
teddy.isClawedBy(teddy);
```

A)
```
"Clawed by a Cat!"
"Clawed by a Kitten!"
"Clawed by a Kitten!"
```

B)
```
"Clawed by a Cat!"
"Clawed by a Cat!"
"Clawed by a Kitten!"
```

Quiz

```
class Cat {
 public void isClawedBy(Kitten c) {
  System.out.println(“Clawed by a Cat!”);
}}
class Kitten extends Cat {
 public void isClawedBy(Cat k) {
  System.out.println(“Clawed by a Kitten!”);
}}
Cat gypsy = new Cat();
Cat spike = new Kitten();
Kitten teddy = new Kitten();
gypsy.isClawedBy(teddy);
spike.isClawedBy(teddy);
teddy.isClawedBy(teddy);
```

A)
```
“Clawed by a Cat!”
“Clawed by a Cat!”
“Clawed by a Kitten!”
```

B)
```
“Clawed by a Cat!”
“Clawed by a Cat!”
“Clawed by a Cat!”
```

Quiz

```
class Cat {
 public void isClawedBy(Cat c, Kitten k) {
  System.out.println("Clawed by a Cat!");
}}
class Kitten extends Cat {
 public void isClawedBy(Kitten k, Cat c) {
  System.out.println("Clawed by a Kitten!");
}}
Cat gypsy = new Cat();
Cat spike = new Kitten();
Kitten teddy = new Kitten();
gypsy.isClawedBy(spike,teddy);
spike.isClawedBy(teddy,spike);
teddy.isClawedBy(teddy,teddy);
```

A)
```
"Clawed by a Cat!"
error
error
```

B)
```
"Clawed by a Cat!"
"Clawed by a Kitten!"
"Clawed by a Kitten!"
```

Dynamic dispatch

- There's more:
  - Access control
  - Static methods
  - Recompilation
  - **`super`**
  - …
- See the Java spec, etc.