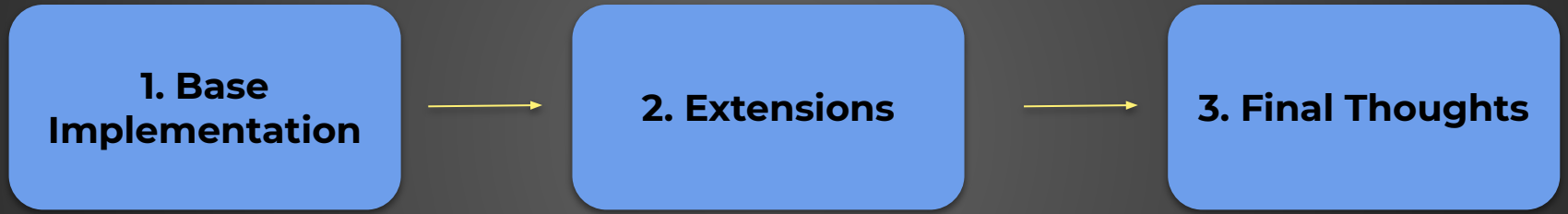
Multiple Dispatch for Java

Francisco Fialho - 98843

Miguel Marcelino - 98684

Group 28

Outline



Problem

- Solve Dynamic Dispatch in Java
  - Using dynamic dispatch not only for the receiver but also for arguments
- Finding most specific method available and removing unexpected results

Algorithm

Algorithm

- **`invoke`**
  - Parses **`args`** into a list for search
  - Filters methods with less or more arguments than specified
  - Calls **`findBestMethod`** to find the most specific method
- **`findBestMethod`** finds most specific method in 2 steps:
  1. Filter out methods where parameters don't match
  2. Find the most specific method amongst filtered methods

Invoke Function

```
public static Object invoke(Object receiver, String name, Object... args) {
        try {
            ArrayList<Object> objects =
                  Arrays.stream(args).collect(Collectors.toCollection(ArrayList::new));
            Class classType = receiver.getClass();

            ArrayList<Method> methodList = Arrays.stream(classType.getMethods())
                    .filter(method -> method.getName().equals(name) &&
                            method.getParameterCount() == objects.size())
                    .collect(Collectors.toCollection(ArrayList::new));

            Method method = findBestMethod(methodList, objects);
            return method.invoke(receiver, args);
        } catch (IllegalAccessException | InvocationTargetException |
                NoSuchMethodException e) {
            throw new RuntimeException(e.getMessage());
        }
}
```

Parse `args`

Filter methods

Call `findBestMethod`

Algorithm - 1

```
static Method findBestMethod(ArrayList<Method> methodList, ArrayList<Object> objects)
            throws NoSuchMethodException {
      List<Method> matchingMethods = methodList.stream()
            .filter(method -> {
                  Class[] parameterTypes = method.getParameterTypes();
                  for (int i = 0; i < objects.size(); i++) {
                        if (!parameterTypes[i].isAssignableFrom(objects.get(i).getClass()))
                              return false;
                  }
                  return true;
            }).collect(Collectors.toList());

      if (matchingMethods.isEmpty()) throw new NoSuchMethodException();
      Method best = matchingMethods.get(0);

      ...
```

Filter methods where parameters are not a match

Algorithm - 2

```
...
for (Method method : matchingMethods) {
      boolean isBest = true;
      boolean isMoreSpecific = false;
      Class[] parameterTypes = method.getParameterTypes();
      Class[] currBestMethodParams = best.getParameterTypes();
      for (int i = 0; i < objects.size() && isBest && !isMoreSpecific; i++) {
            if (currBestMethodParams[i].isAssignableFrom(parameterTypes[i])) {
                  if (currBestMethodParams[i] != parameterTypes[i])
                        isMoreSpecific = true;
            } else {
                  isBest = false;
            }
      }

      if (isMoreSpecific) best = method;
  }
 return best;
}
```

Find the most specific method

Extensions

Dealing with Boxing and Unboxing

- Input array was added to **`invoke`** to specify elements that need to be unboxed
- **`findBestMethod`** needs to convert an object to a primitive type if its position is specified in the input array
- In our implementation:
  - `someMethod(int)` → `someMethod(int)` or `someMethod(Integer)`
  - `someMethod(Integer)` → `someMethod(Integer)`

Invoke Function

```
public static Object invoke(Object receiver, String name, Integer[] primitiveObjPositions, Object... args) {
        try {
            ArrayList<Object> objects =
                  Arrays.stream(args).collect(Collectors.toCollection(ArrayList::new));
            ArrayList<Integer> primitivePositions =
                  Arrays.stream(primitiveObjPositions);
            Class classType = receiver.getClass();

            ArrayList<Method> methodList = Arrays.stream(classType.getMethods())
                    .filter(method -> method.getName().equals(name) &&
                            method.getParameterCount() == objects.size())
                    .collect(Collectors.toCollection(ArrayList::new));

            Method method = findBestMethod(methodList, objects, primitivePositions);
            return method.invoke(receiver, args);
        } catch (IllegalAccessException | InvocationTargetException |
                NoSuchMethodException e) {
            throw new RuntimeException(e.getMessage());
        }
}
```

New parsing

Adds `primitivePositions`

Algorithm Boxing/Unboxing - 1

```
static Method findBestMethod(ArrayList<Method> methodList, ArrayList<Object> objects,
                             ArrayList<Integer> primitivePositions) throws NoSuchMethodException {
        List<Method> matchingMethods = methodList.stream().filter(method -> {
            Class[] parameterTypes = method.getParameterTypes();
            for (int i = 0; i < objects.size(); i++) {
                Class objectClass = objects.get(i).getClass();
                if (primitivePositions.contains(i)) {
                    Class primitiveClass = getPrimitiveType(objectClass);
                    if (parameterTypes[i].isPrimitive()) {
                        if (!parameterTypes[i].isAssignableFrom(primitiveClass))
                            return false;
                    } else {
                        if (!parameterTypes[i].isAssignableFrom(objectClass))
                            return false;
                    }
                } else {
                    if (!parameterTypes[i].isAssignableFrom(objectClass))
                        return false;
                }
            }
            return true;
        }).collect(Collectors.toList());
```

New condition verification

Algorithm Boxing/Unboxing - 2

```
if (matchingMethods.isEmpty()) throw new NoSuchMethodException();
Method best = matchingMethods.get(0);

for (Method method : matchingMethods) {
      boolean isBest = true;
      boolean isMoreSpecific = false;
      Class[] parameterTypes = method.getParameterTypes();
      Class[] currBestMethodParams = best.getParameterTypes();
      for (int i = 0; i < objects.size() && isBest && !isMoreSpecific; i++) {
            if (primitivePositions.contains(i)) {
                  if (!currBestMethodParams[i].isPrimitive()) {
                        if (parameterTypes[i].isPrimitive())
                              isMoreSpecific = true;
                  }
```

New condition verification

Example:
currBestMethodParams[i] = Integer
parameterTypes[i] = int

Algorithm Boxing/Unboxing - 3

```
                } else {
                      if (currBestMethodParams[i].isAssignableFrom(parameterTypes[i])) {
                            if (currBestMethodParams[i] != parameterTypes[i])
                                  isMoreSpecific = true;
                            } else {
                                  isBest = false;
                            }
                      }
                }

                if (isMoreSpecific) best = method;
      }

      return best;
}
```

Dealing with Interface hierarchy

- Requires another condition
  - Find most specific Interface
- In our implementation:

```
public class TesterClass extends Test implements ITest1, ITest2 → ITest1
public class TesterClass extends Test implements ITest2, ITest1 → ITest2
```

Algorithm Interfaces - 1

```
...
if (matchingMethods.isEmpty()) throw new NoSuchMethodException();
Method best = matchingMethods.get(0);

for (Method method : matchingMethods) {
      boolean isBest = true;
      boolean isMoreSpecific = false;
      Class[] parameterTypes = method.getParameterTypes();
      Class[] currBestMethodParams = best.getParameterTypes();
      for (int i = 0; i < objects.size() && isBest && !isMoreSpecific; i++) {
            if (currBestMethodParams[i].isAssignableFrom(parameterTypes[i])) {
                  if (currBestMethodParams[i] != parameterTypes[i])
                        isMoreSpecific = true;
            } else {
                  ...
```

Algorithm Interfaces - 2

New Conditions

```
            ...
            if (currBestMethodParams[i].isInterface() && parameterTypes[i].isInterface()) {
                  ArrayList<Class> interfaces =
                        Arrays.stream(objects.get(i).getClass().getInterfaces())
                              .collect(Collectors.toCollection(ArrayList::new));
                  if (!interfaces.isEmpty()) {
                        if (interfaces.indexOf(currBestMethodParams[i]) <
                                    interfaces.indexOf(parameterTypes[i]))
                              isBest = false;
                        else
                              isMoreSpecific = true;
                  }
            } else {
                  isBest = false;
            }
      }
   }
   if (isMoreSpecific) best = method;
...
}
```

Final Thoughts

Final Thoughts

- These are complex topics
- Our algorithm tries to maintain as many java properties as possible
- Interface implementation could get more specific

Multiple Dispatch for Java

Francisco Fialho - 98843

Miguel Marcelino - 98684

Group 28