What You Should Already Know About Java

Felix Herrmann

November 30, 2016

Übersicht

Common Mistakes

Useful Features

Common Mistakes

String Comparison

- `==` compares objects by reference (identity)
- Strings should not be compared with `==`
  - unless you *really* know what you are doing
  - and then, please write a comment
  - or give the function doing black magic a very descriptive name
- It might seem as if it works, but it doesn't

Strings Comparison

```
"x" == "x"
```

```
new String("x") == new String("x")
```

```
"x" + "x" == "x" + "x"
```

```
repeat("x", 2) == repeat("x", 2)
```

Strings Comparison

true `"x" == "x"`

false `new String("x") == new String("x")`

true `"x" + "x" == "x" + "x"`

false `repeat("x", 2) == repeat("x", 2)`

- This is because string constants are `.intern()` ed
- Use `.equals(other)` instead

```
"x".equals("x")
new String("x").equals(String("x"))
("x" + "x").equals("x" + "x")
repeat("x", 2).equals(repeat("x", 2))
```

Strings Comparison

true `"x" == "x"`

false `new String("x") == new String("x")`

true `"x" + "x" == "x" + "x"`

false `repeat("x", 2) == repeat("x", 2)`

- This is because string constants are `.intern()` ed
- Use `.equals(other)` instead

true `"x".equals("x")`

true `new String("x").equals(String("x"))`

true `("x" + "x").equals("x" + "x")`

true `repeat("x", 2).equals(repeat("x", 2))`

HashCode And Equals

- Always override both or neither
- If they are equal, they need to have the same hash
- If they have the same hash, they *might* be equal
- The type of the parameter of `equals` has to be `Object`
- Very hard in a class hierarchy, because equality needs to be
  - Reflexive
  - ***Symmetric***
  - ***Transitive***
  - Consistent
  - Not equal to `null`

HashCode And Equals: Object Type

Wrong:

```
private class Equal {
  public boolean equals(Equal other) {
    return true;
  }
}
```

```
List<Equal> nots = Arrays.asList(new Equal());
System.out.println(nots.contains(new Equal()));
// prints false
```

HashCode And Equals: Object Type

- Right:

```
private class Equal0v {
  @Override
  public boolean equals(Object other) {
    return true;
  }
}
```

- Always use override. Wrong version with `@Override` does not compile.

HashCode And Equals: Symmetrical

```
class Point {
  private final int x;
  private final int y;

  @Override
  public boolean equals(Object obj) {
    if (!(obj instanceof Point))
      return false;
    Point o = (Point) obj;
    return x == o.x && y == o.y;
  }

  /* more stuff ... */
```

HashCode And Equals: Symmetrical

```
class ColorPoint extends Point {
  private final int c;

  @Override
  public boolean equals(Object obj) {
    if (!(obj instanceof ColorPoint))
      return false;
    ColorPoint o = (ColorPoint) obj;
    return c == o.c && super.equals(o);
  }

  /* more stuff ... */
```

HashCode And Equals: Symmetrical

Not symmetrical:

```
Point p1 = new Point(1, 0);
ColorPoint p2 = new ColorPoint(1, 0, 1);

System.out.println(p1.equals(p2)); // true
System.out.println(p2.equals(p1)); // false
```

HashCode And Equals: Transitive

Fix symmetry:

```
public boolean equals(Object obj) {
  if (obj instanceof ColorPointSym) {
    ColorPointSym o = (ColorPointSym) obj;
    return c == o.c && super.equals(o);
  } else if (obj instanceof Point) {
    return super.equals(obj);
  }
  return false;
}
```

HashCode And Equals: Transitive

- Not transitive:

```
  ColorPointSym p1 = new ColorPointSym(1, 0, 1);
  Point p2 = new Point(1, 0);
  ColorPointSym p3 = new ColorPointSym(1, 0, 2);

  System.out.println(p1.equals(p2)); // true
  System.out.println(p2.equals(p1)); // true
  System.out.println(p2.equals(p3)); // true
  System.out.println(p1.equals(p3)); // false
}
```

- No easy solution
- Maybe go with composition instead of inheritance

Resources Need To Be Freed

```
for (int i = 0; i < 10000000; i++) {
    Files.newOutputStream(Paths.get("f/out" + i));
}
```

- Throws an `IOException` because there are too many open files
- Always close resources

Resources Need To Be Freed

```
OutputStream out =
↪  Files.newOutputStream(Paths.get("f/out" + i));
doSomethingWithOut(out);
out.close();
```

- Seems simple, but isn't
- Leaves files open if `doSomethingWithOut` throws an exception

Resources Need To Be Freed

- Try-With-Resource statement to the rescue!
- Closes all resources at end of block
- Catches all the corner cases, even for multiple resources

```
try (R1 r1 = getR1(); R2 r2 = getR2()) {
  doSomethingWithR1(r1);
  doSomethingWithR2(r2);
}
```

```
try (OutputStream out =
↪   Files.newOutputStream(Paths.get("f/out" + i))) {
    doSomethingWithOut(out);
}
```

Useful Features

Use The Enhanced For Loop

- Don't do this:

```
for (int i = 0; i < list.size(); i++)
  doSomethingWith(list.get(i));
```

Use The Enhanced For Loop

- Don't do this:

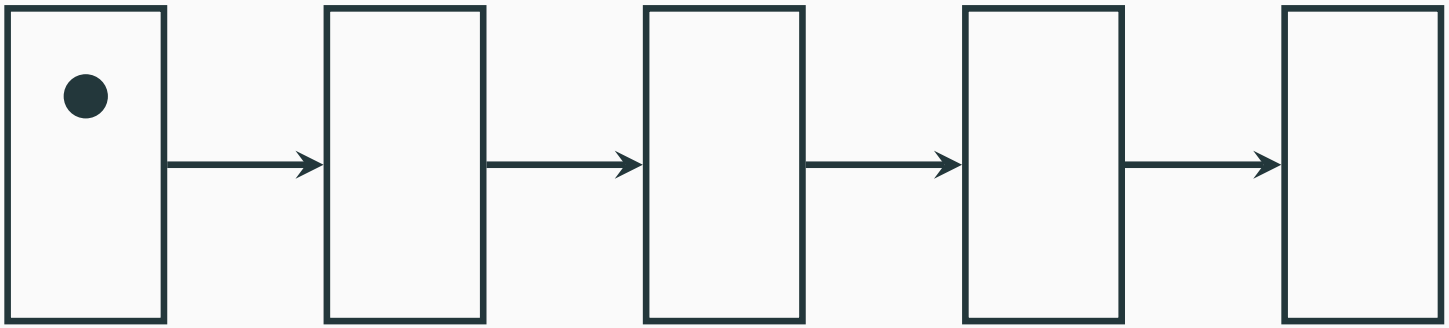
```
for (int i = 0; i < list.size(); i++)
  doSomethingWith(list.get(i));
```

- The index is not needed apart from accessing the element
- Might be very slow
- Easy to fuck up

```
for (int i = 0; i <= list.size(); i++)
  doSomethingWith(list.get(i));
```

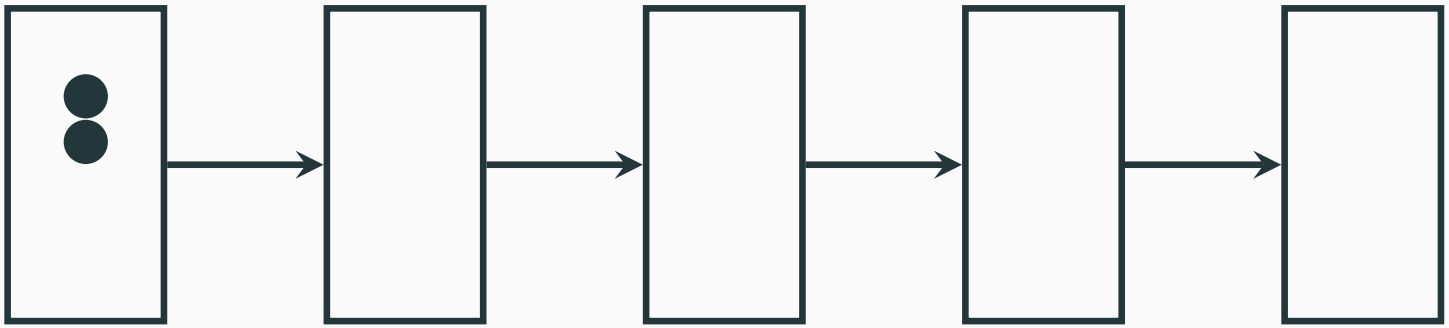
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`



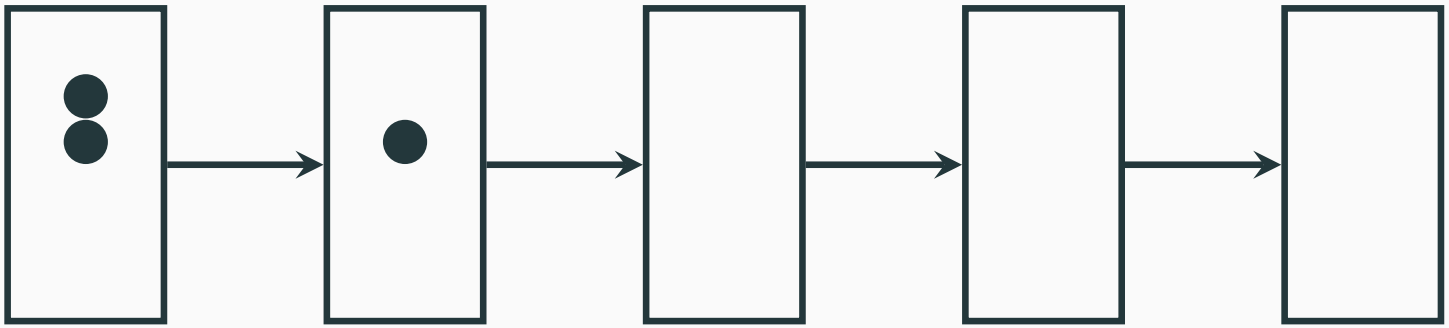
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`



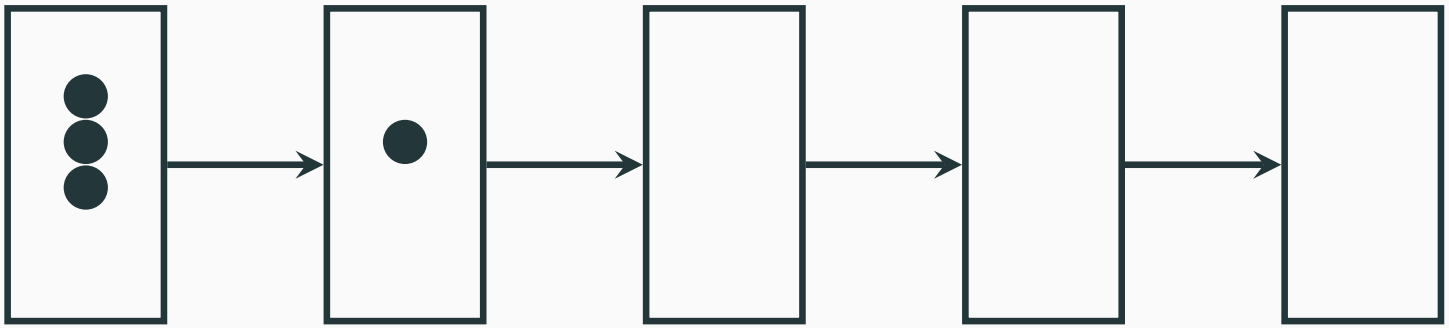
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`



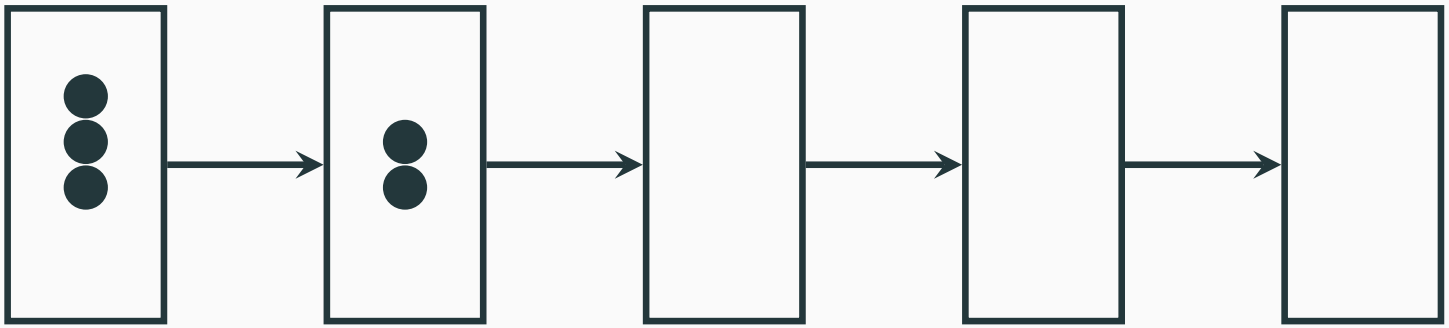
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`



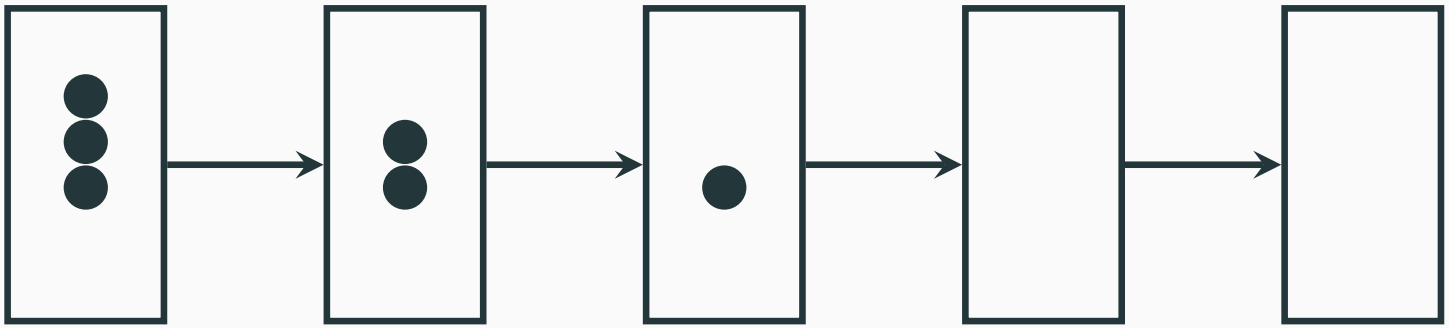
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`



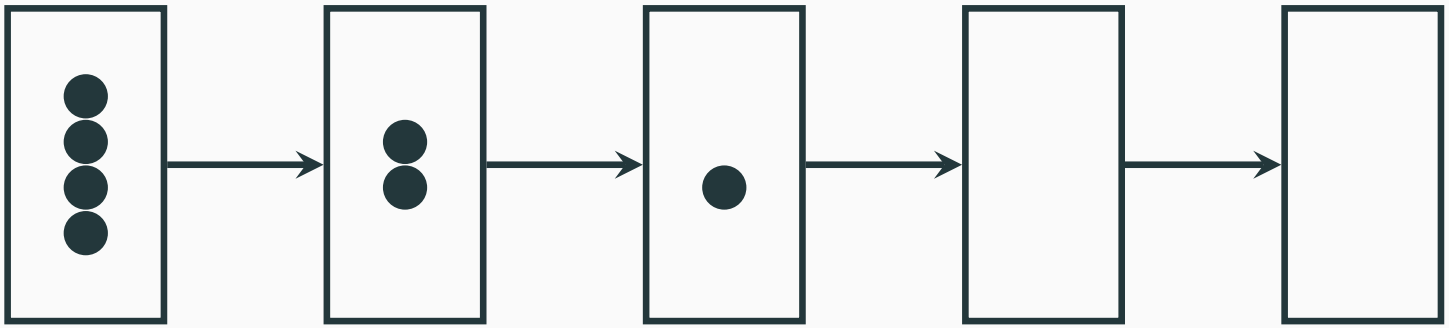
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`



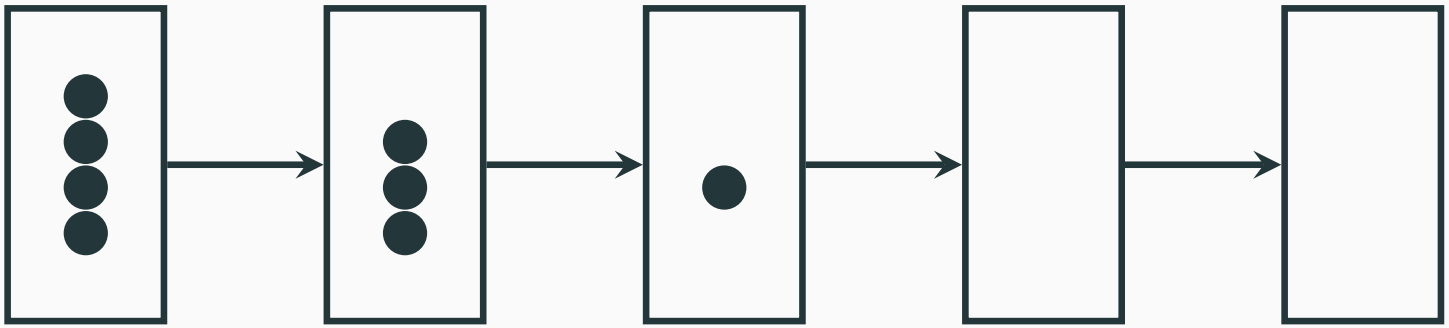
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`
- `get(3)`



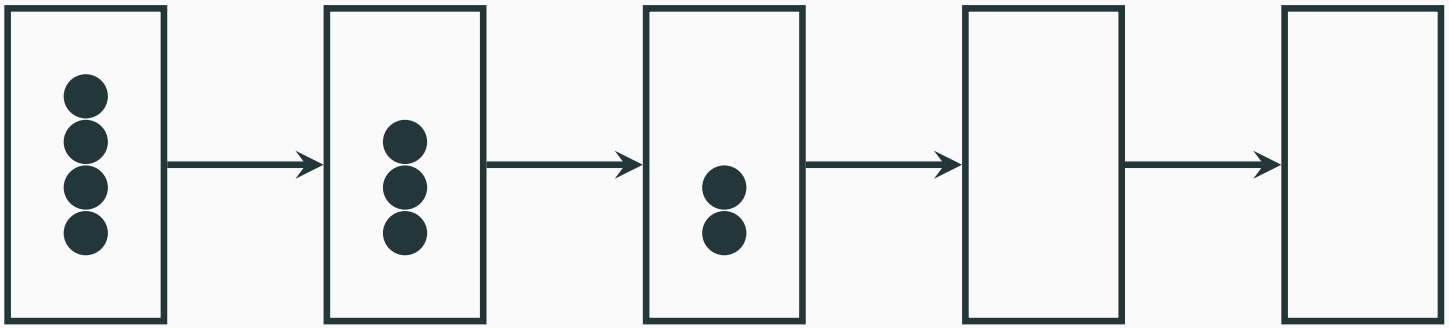
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`
- `get(3)`



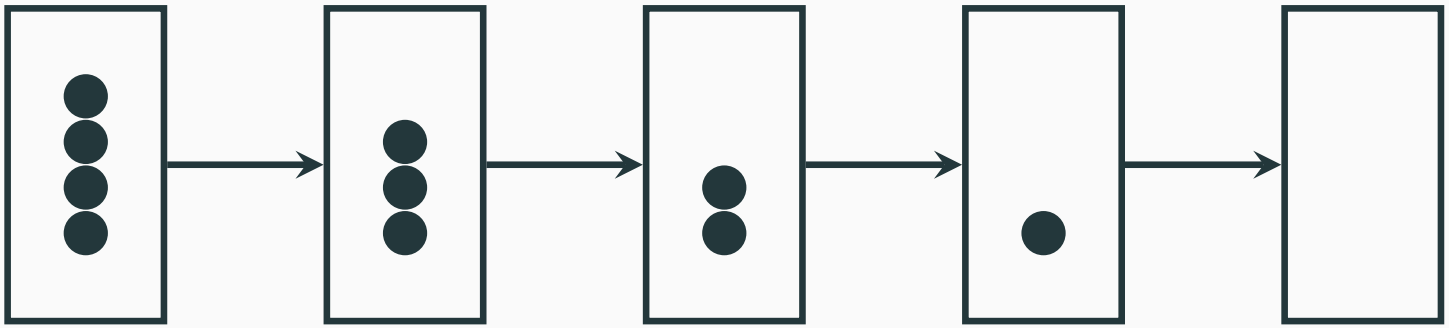
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`
- `get(3)`



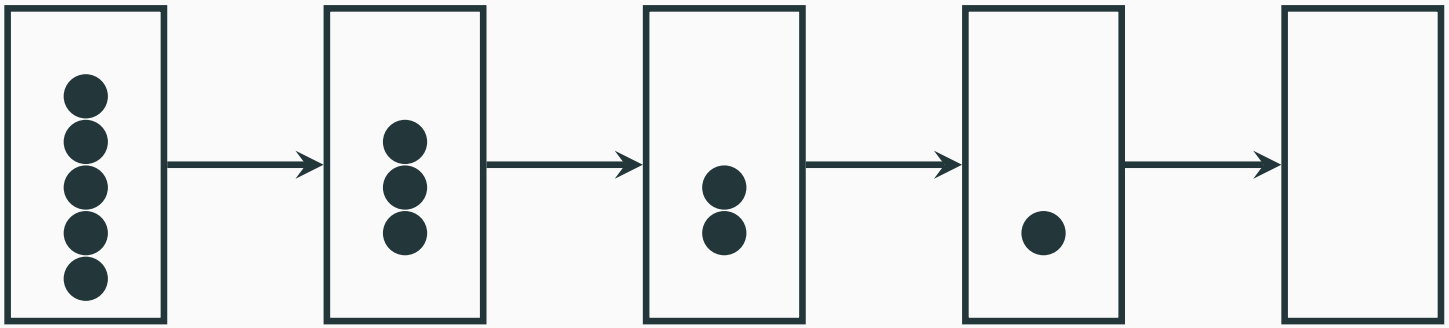
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`
- `get(3)`



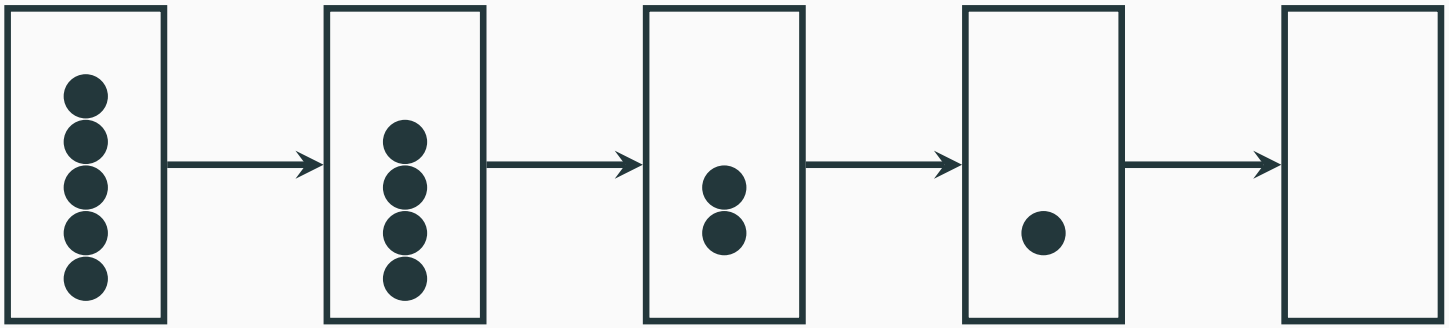
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`
- `get(3)`
- `get(4)`



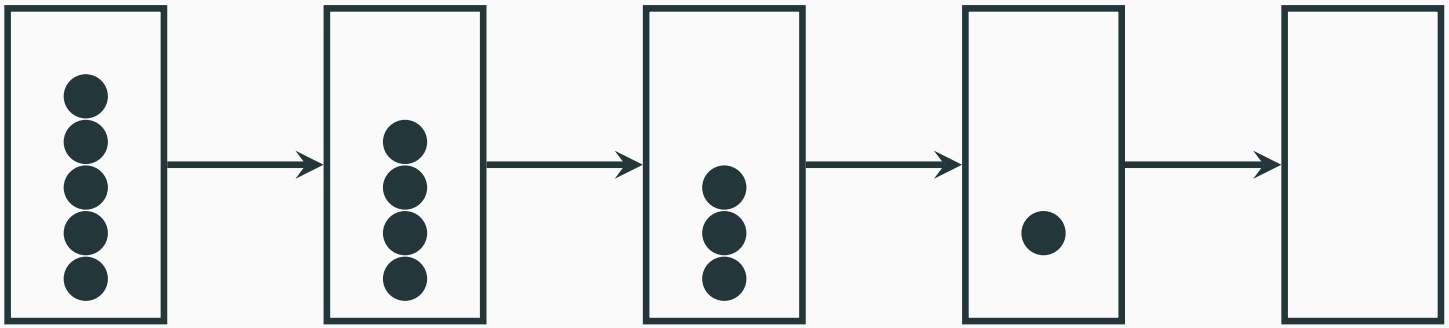
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`
- `get(3)`
- `get(4)`



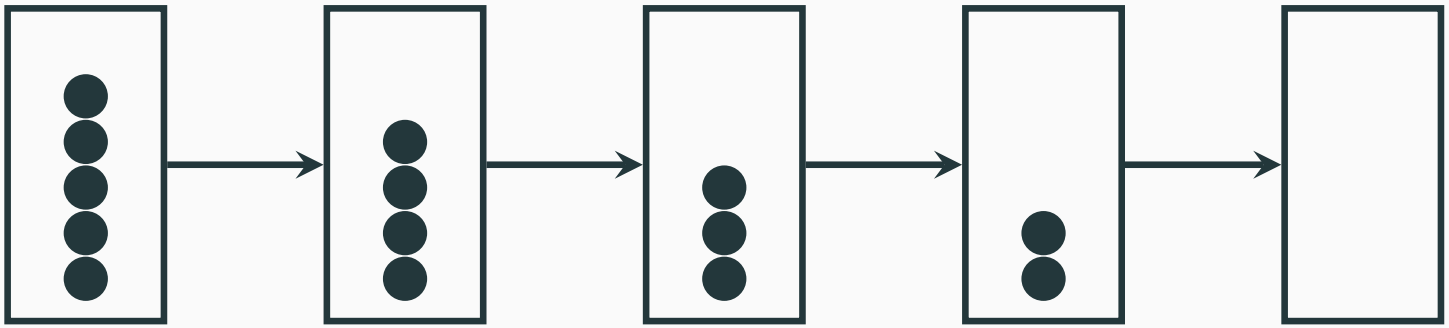
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`
- `get(3)`
- `get(4)`



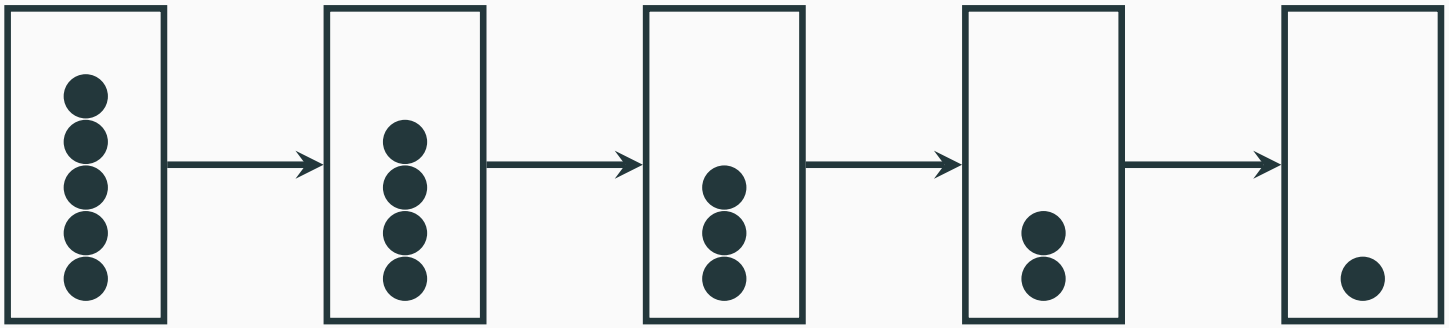
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`
- `get(3)`
- `get(4)`



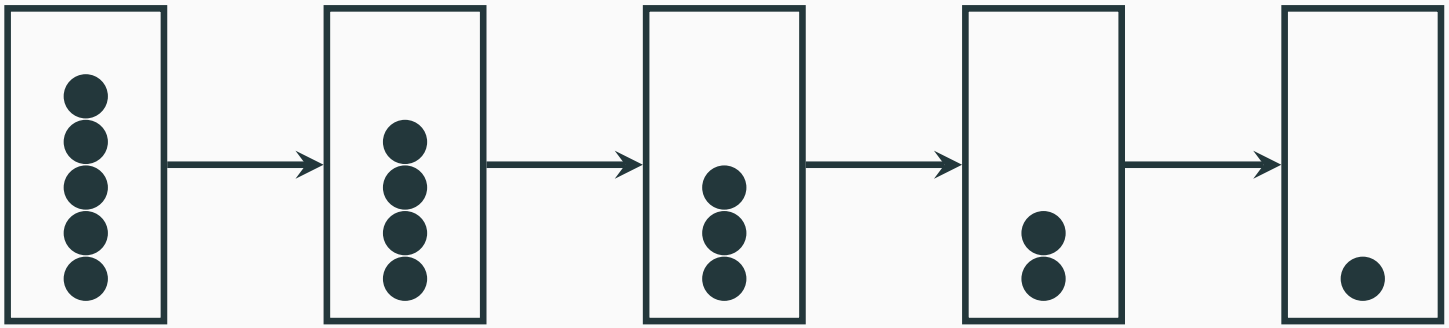
Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`
- `get(3)`
- `get(4)`



Use The Enhanced For Loop

- Slow example: linked list
- `get(0)`
- `get(1)`
- `get(2)`
- `get(3)`
- `get(4)`



Use The Enhanced For Loop

- Alternative:

```
final Iterator<Integer> it = list.iterator();
while (it.hasNext()) {
  doSomethingWith(it.next());
}
```

- Is faster
- But introduces temporary variable
- Still easy to fuck up (calling `next()` inside the loop)

Use The Enhanced For Loop

- Use the enhanced for loop instead

```
for (int i : list)
  doSomethingWith(i);
```

- Is fast
- No temporary variable
- Not as easy to fuck up

Use Objects: HashCode And Equals

- Use `java.util.Objects` for common tasks.
- Simple class:

```
private final String firstName;
private final Optional<String> middleName;
private final String lastName;
private final Optional<Integer> age;
private int cache = 0;
```

Use Objects: HashCode

Instead of this

```
@Override
public int hashCode() {
  final int prime = 31;
  int result = 1;
  result = prime * result + ((age == null) ? 0 :
  ↪  age.hashCode());
  result = prime * result + ((firstName == null) ? 0 :
  ↪  firstName.hashCode());
  result = prime * result + ((lastName == null) ? 0 :
  ↪  lastName.hashCode());
  result = prime * result + ((middleName == null) ? 0 :
  ↪  middleName.hashCode());
  return result;
}
```

Use Objects: HashCode

Use this

```
@Override
public int hashCode() {
  return Objects.hash(firstName, middleName, lastName,
  ↪  age);
}
```

Use Objects: Equals

Instead of this

```
@Override
public boolean equals(Object obj) {
  if (this == obj)
    return true;
  if (obj == null)
    return false;
  if (getClass() != obj.getClass())
    return false;
  PersonOld other = (PersonOld) obj;
  if (age == null) {
    if (other.age != null)
      return false;
  } else if (!age.equals(other.age))
```

Use Objects: Equals

```
  return false;
if (firstName == null) {
  if (other.firstName != null)
    return false;
} else if (!firstName.equals(other.firstName))
  return false;
if (lastName == null) {
  if (other.lastName != null)
    return false;
} else if (!lastName.equals(other.lastName))
  return false;
if (middleName == null) {
  if (other.middleName != null)
    return false;
} else if (!middleName.equals(other.middleName))
```

Use Objects: Equals

```
    return false;
  return true;
}
```

Use Objects: Equals

Use this

```
@Override
public boolean equals(Object obj) {
  if (!(obj instanceof PersonNew))
    return false;

  final PersonNew o = (PersonNew) obj;
  return Objects.equals(firstName, o.firstName) &&
    Objects.equals(middleName, o.middleName) &&
    Objects.equals(lastName, o.lastName) &&
    Objects.equals(age, o.age);
}
```

Use Objects: RequireNonNull

Instead of this

```
class PersonOld {
  private final String name;
  private final int age;

  public PersonOld(String name, int age) {
    if (name == null) {
      throw new NullPointerException("null");
    }
    this.name = name;
    this.age = age;
  }
}
```

Use Objects: RequireNonNull

Use this

```
class PersonNew {
  private final String name;
  private final int age;

  public PersonNew(String name, int age) {
    this.name = Objects.requireNonNull(name, "null");
    this.age = age;
  }
}
```