A Friendly Introduction to Software Documentation

Components of ACCEU

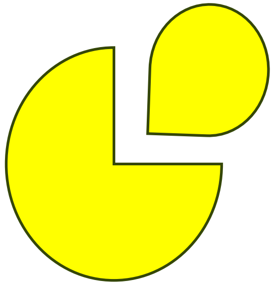
By Arist Bravo

Our Dimension-Based Framework, ACCEU

- We choose the dimension-based framework since it is commonly used (see references) and provides rules-of-thumb for "good documentation"
- After all: most documentation is written without standards!
- We study five (unordered) qualities/dimensions based on common use:

Accuracy

Clarity



Completeness

Ease-of-use

Up-to-dateness

Reference(s): Forward and Lethbridge, Garousi et al., Parnas, Treude et al., Zhi et al.

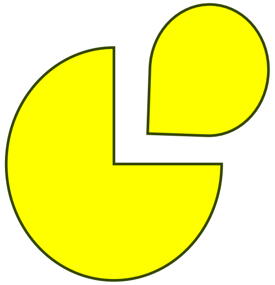
Why ACCEU?

Comment Level

Accuracy

Clarity

Document Level



Completeness

Ease-of-use

Process Level

Up-to-dateness

Accuracy

- Definition: docs which are *true* (i.e. correctly says what the code does) and *relevant*
- Some things are true but trivial; write key ideas (code's purpose), not obvious fluff
- "Misinformation can be more damaging than missing information" (Parnas)

Accuracy

Reference(s): Parnas, Zhi et al.

Clarity

- Definition: docs which can only be interpreted one way; they are not ambiguous
- If docs aren't written clearly, then your code gets viewed in many ways
- Many interpretations → many ways to "use" your code → a dev misuses or breaks code

Clarity

Reference(s): Parnas, Zhi et al.

Accuracy ≠ Clarity

- Having one does not mean having the other!
- EX from CSC207: online dating app, with a use case of figuring out whether two users would like each other based on preferences
- Each user has these aspects:
  - Gender (**M**ale, **F**emale, **N**on-binary)
  - Sexuality (**H**eterosexual, **L**esbian, **G**ay, **B**isexual)

Specifications

User's Sexuality	Preferred Gender	Preconditions
H(eterosexual)	User's opposite gender, or N if the user is N	N/A
L(esbian)	F	Only F users can be L
G(ay)	M	Only M users can be G
B(isexual)	Any	N/A

Reference(s): Parnas, Zhi et al.

Case Study: Compatibility of Users

```
// A constant outside any function
HashMap<String, String> desiredGender =
  new HashMap<>();
desiredGender.put("B", "MFN");
desiredGender.put("L", "F");
desiredGender.put("G", "M");
desiredGender.put("HM", "F");
desiredGender.put("HF", "M");
desiredGender.put("HN", "N");
```

```
public boolean genderSexMatches (String u1Gender,
String u1Sex, String u2Gender, String u2Sex) {
  String u1Wants;
  if (Objects.equals(u1Sex, "H")) {
    u1Wants = desiredGender.get("H" + u1Gender);
  } else {
    u1Wants = desiredGender.get(u1Sex);
  }
  String u2Wants;
  if (Objects.equals(u2Sex, "H")) {
    u2Wants = desiredGender.get("H" + u2Gender);
  } else {
    u2Wants = desiredGender.get(u2Sex);
  }
  return (u2Wants.contains(u1Gender) &&
u1Wants.contains(u2Gender));
}
```

Not Accurate (Truth Issue), Not Clear

```
  // Get sexuality
  String u1Wants;
  if (Objects.equals(u1Sex, "H")) {
    u1Wants = desiredGender.get("H" + u1Gender);
  } else {
    u1Wants = desiredGender.get(u1Sex);
  }
  String u2Wants;
  if (Objects.equals(u2Sex, "H")) {
    u2Wants = desiredGender.get("H" + u2Gender);
  } else {
    u2Wants = desiredGender.get(u2Sex);
  }

  // Return whether sexualities match
  return (u2Wants.contains(u1Gender) &&
u1Wants.contains(u2Gender));
}
```

Not Accurate (Relevance Issue), Not Clear

```
  // Get genders from the hash map of desired genders
  String u1Wants;
  if (Objects.equals(u1Sex, "H")) {
    u1Wants = desiredGender.get("H" + u1Gender);
  } else {
    u1Wants = desiredGender.get(u1Sex);
  }
  String u2Wants;
  if (Objects.equals(u2Sex, "H")) {
    u2Wants = desiredGender.get("H" + u2Gender);
  } else {
    u2Wants = desiredGender.get(u2Sex);
  }

  // Return whether the users are compatible
  return (u2Wants.contains(u1Gender) && u1Wants.contains(u2Gender));
}
```

Clear, But Not Accurate

```
// For each user, get a gender from the hash map of desired genders
String u1Wants;
if (Objects.equals(u1Sex, "H")) {
  u1Wants = desiredGender.get("H" + u1Gender);
} else {
  u1Wants = desiredGender.get(u1Sex);
}
String u2Wants;
if (Objects.equals(u2Sex, "H")) {
  u2Wants = desiredGender.get("H" + u2Gender);
} else {
  u2Wants = desiredGender.get(u2Sex);
}
...
```

Accurate, But Not Clear

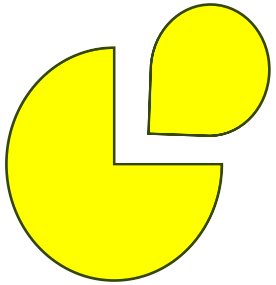
```
 // For each user, use the hash map to get the gender they desire from
a partner
 String u1Wants;
 if (Objects.equals(u1Sex, "H")) {
  u1Wants = desiredGender.get("H" + u1Gender);
 } else {
  u1Wants = desiredGender.get(u1Sex);
 }
 String u2Wants;
 if (Objects.equals(u2Sex, "H")) {
  u2Wants = desiredGender.get("H" + u2Gender);
 } else {
  u2Wants = desiredGender.get(u2Sex);
 }
 ...
```

Accurate and Clear

```
  // For each user, get the gender (that the user desires in a partner)
by referencing the hash map
  // if a user is heterosexual, then use that sexuality and the user's
gender to determine the desired gender; else, only use the user's
sexuality
  String u1Wants;
  if (Objects.equals(u1Sex, "H")) {
    u1Wants = desiredGender.get("H" + u1Gender);
  } else {
    u1Wants = desiredGender.get(u1Sex);
  }
  String u2Wants;
  if (Objects.equals(u2Sex, "H")) {
    u2Wants = desiredGender.get("H" + u2Gender);
  } else {
    u2Wants = desiredGender.get(u2Sex);
  }
  ...
```

Completeness

- Definition: docs provide all the info you need to understand how the code works
- Describes behaviour of a program (depending on inputs) + data used + data format
- Good variable names and format can provide info too, reducing the need for comments



Completeness

Reference(s): Parnas, Zhi et al.

Ease-of-Use

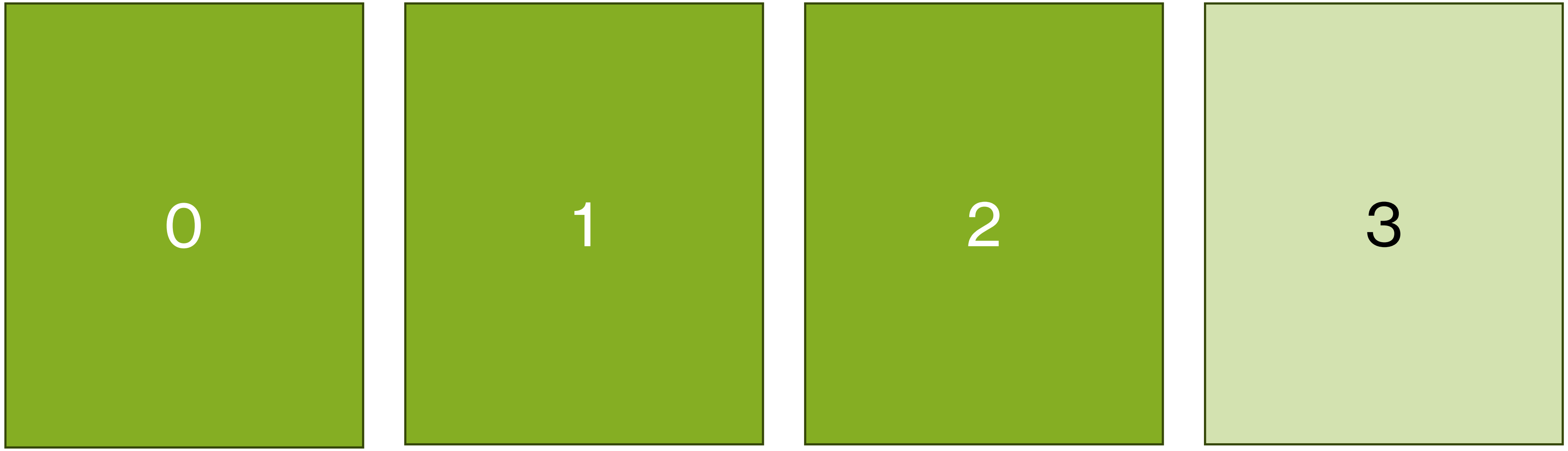
- Definition: docs are easy to navigate; you can easily find the info you need
- Uniformly organized docs (same format for function headers, program headers, etc.)
- Don't have to go all over the doc to find info about one object, entity, or idea

Ease-of-use

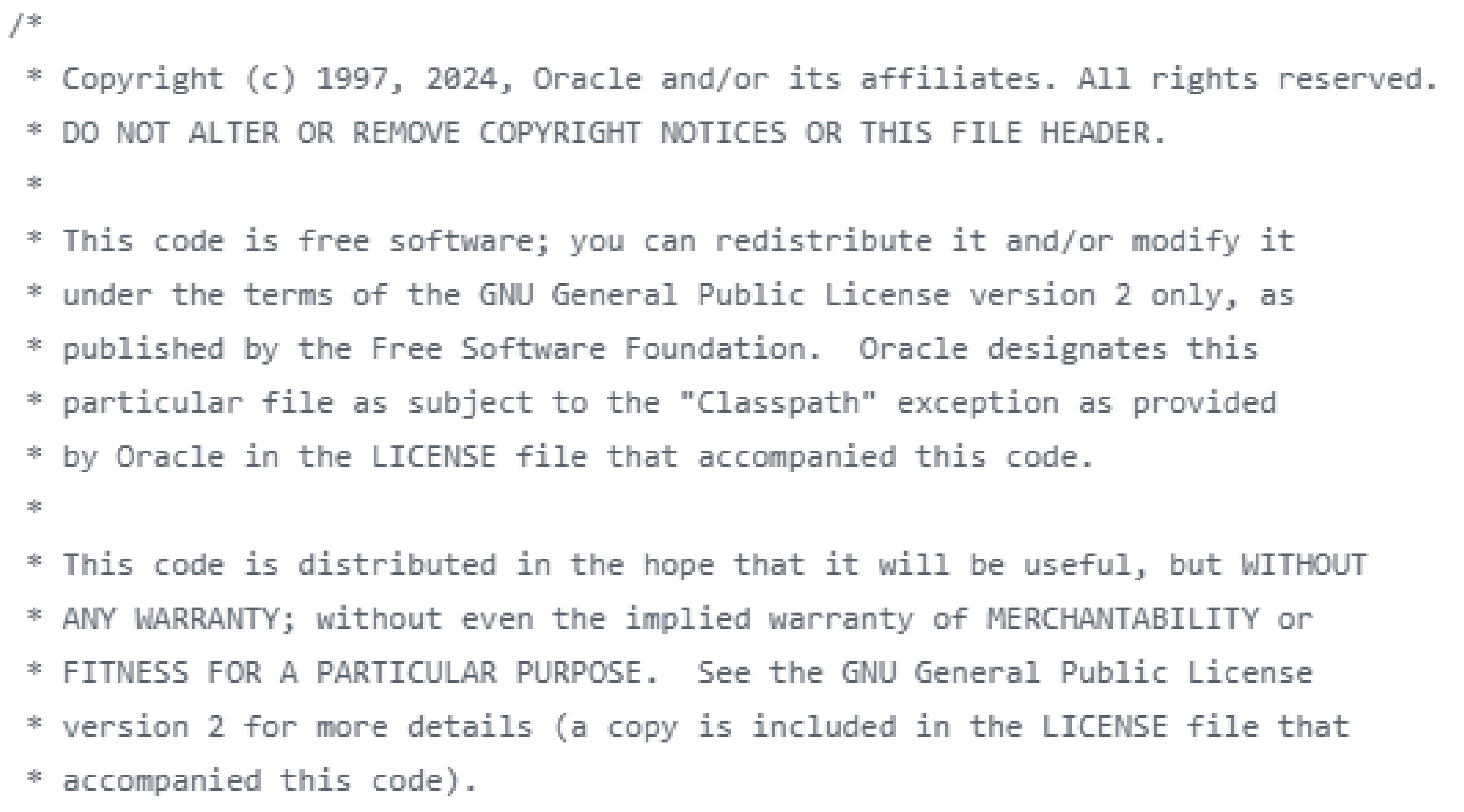
Reference(s): Parnas, Zhi et al.

Case Study: ArrayList.java

- ArrayList.java is the Java implementation of the ArrayList data structure
- Has examples of good documentation

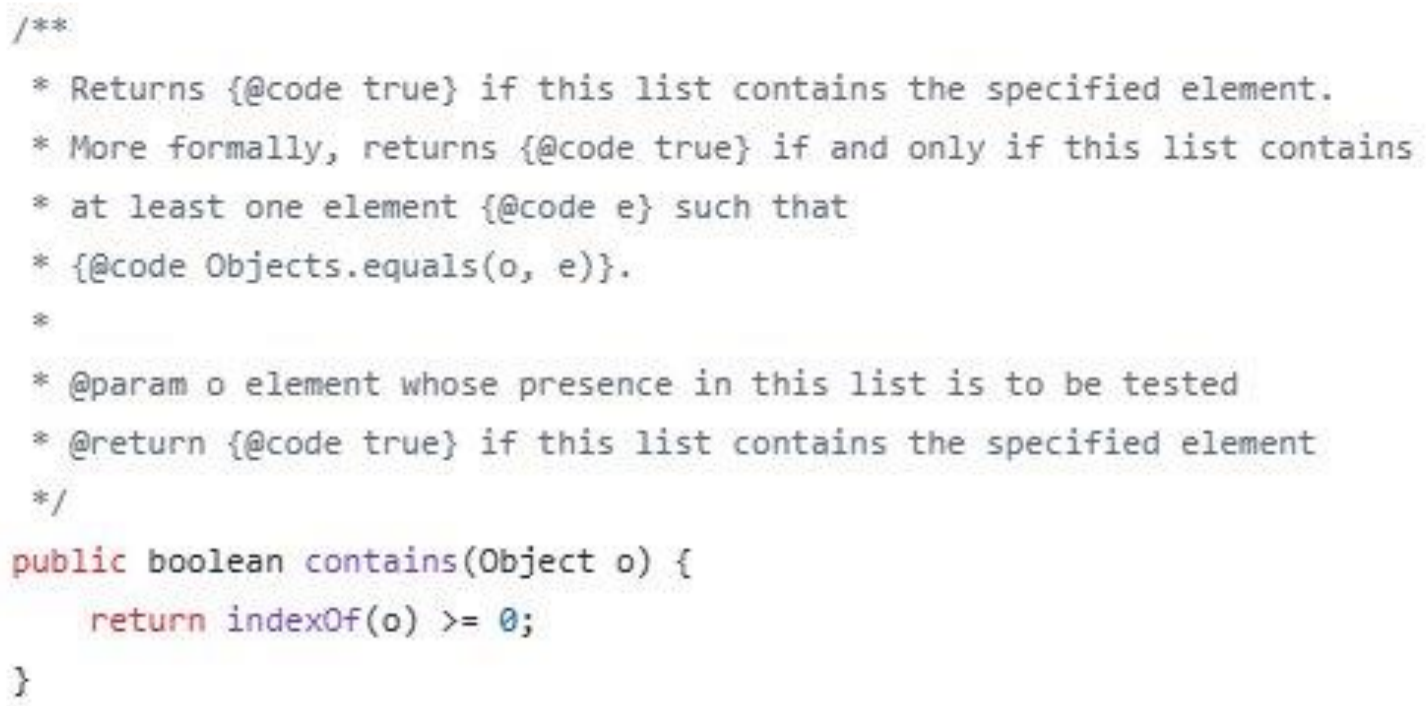


ArrayList.java: Program Header



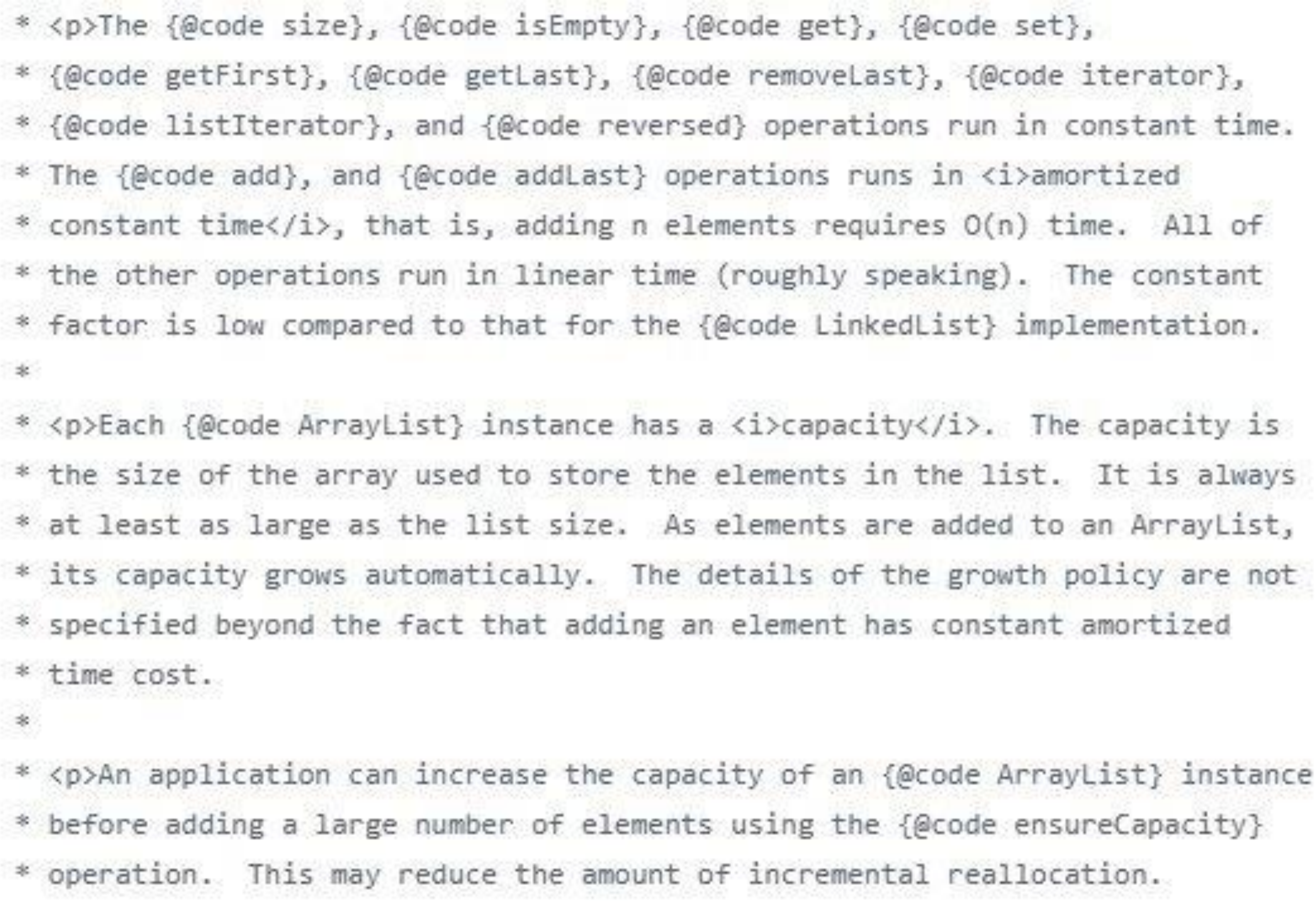
```
/*
 * Copyright (c) 1997, 2024, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Oracle designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Oracle in the LICENSE file that accompanied this code.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
```

ArrayList.java: Function Header Example



```java
/**
 * Returns {@code true} if this list contains the specified element.
 * More formally, returns {@code true} if and only if this list contains
 * at least one element {@code e} such that
 * {@code Objects.equals(o, e)}.
 *
 * @param o element whose presence in this list is to be tested
 * @return {@code true} if this list contains the specified element
 */
public boolean contains(Object o) {
    return indexOf(o) >= 0;
}
```

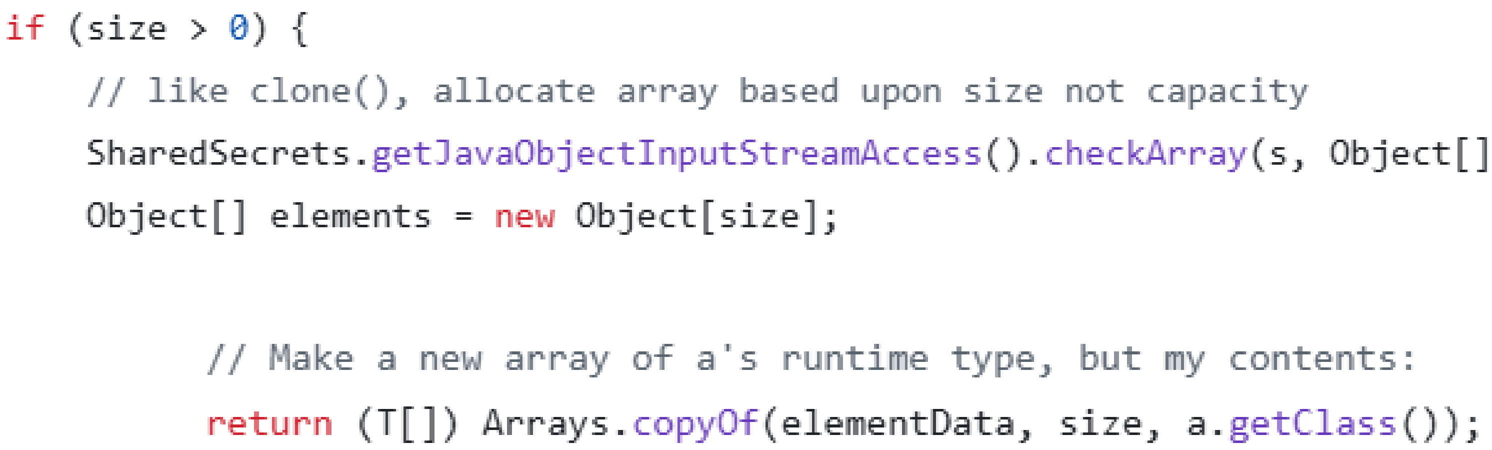
ArrayList.java: Class Header Snippet



```
* <p>The {@code size}, {@code isEmpty}, {@code get}, {@code set},
* {@code getFirst}, {@code getLast}, {@code removeLast}, {@code iterator},
* {@code listIterator}, and {@code reversed} operations run in constant time.
* The {@code add}, and {@code addLast} operations runs in <i>amortized
* constant time</i>, that is, adding n elements requires O(n) time.  All of
* the other operations run in linear time (roughly speaking).  The constant
* factor is low compared to that for the {@code LinkedList} implementation.
*
* <p>Each {@code ArrayList} instance has a <i>capacity</i>.  The capacity is
* the size of the array used to store the elements in the list.  It is always
* at least as large as the list size.  As elements are added to an ArrayList,
* its capacity grows automatically.  The details of the growth policy are not
* specified beyond the fact that adding an element has constant amortized
* time cost.
*
* <p>An application can increase the capacity of an {@code ArrayList} instance
* before adding a large number of elements using the {@code ensureCapacity}
* operation.  This may reduce the amount of incremental reallocation.
```

Side Note: Inline Comments?

- If you study ArrayList.java, you see inline comments occasionally occur
- They're used to justify the reasons for doing something – why implement in this particular way? What exactly is going on?
- Clarifies the code's purpose when it's not obvious → more completeness



```
if (size > 0) {
    // like clone(), allocate array based upon size not capacity
    SharedSecrets.getJavaObjectInputStreamAccess().checkArray(s, Object[].class, size);
    Object[] elements = new Object[size];


          // Make a new array of a's runtime type, but my contents:
          return (T[]) Arrays.copyOf(elementData, size, a.getClass());
```

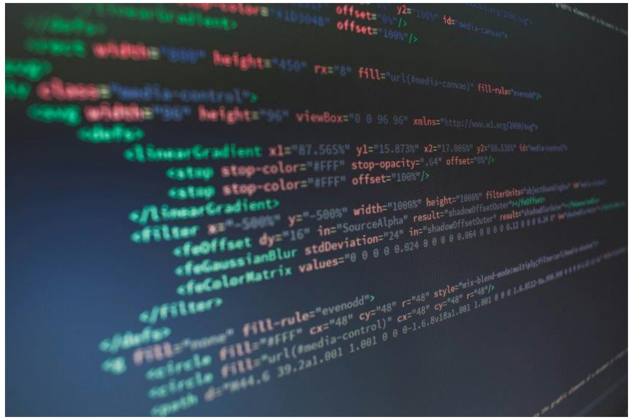
Up-to-dateness

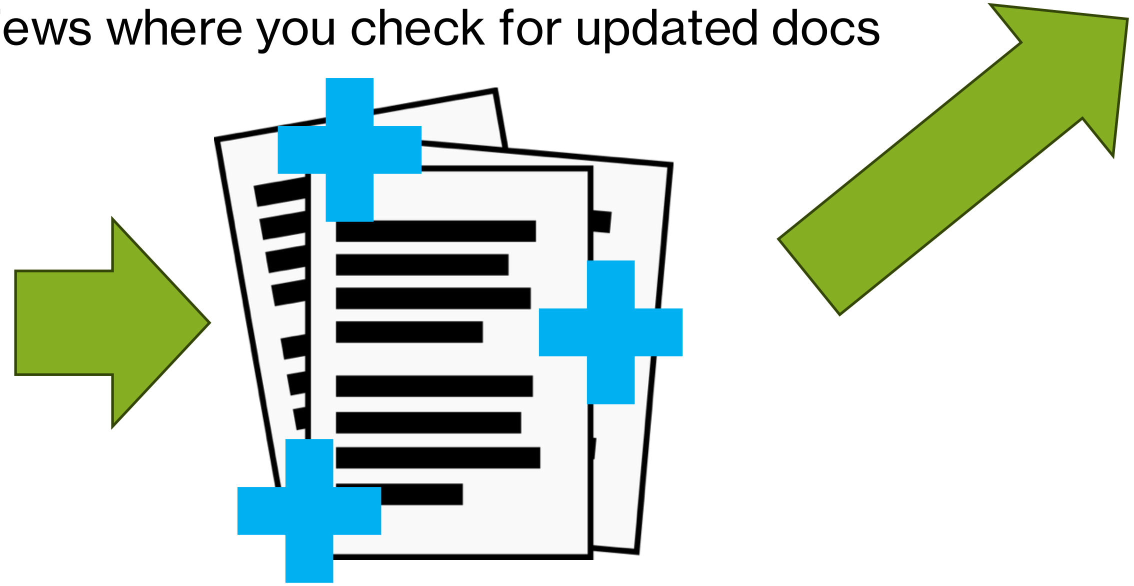
- Definition: docs are kept up-to-date and accurate, as new versions of code are released
- Includes (but not limited to) updating headers, comments, and even README files
- Requires good practices put into your workflow

Up-to-dateness

Best Practices for Updating Docs

- Keep docs in one place with version control (same repo)
- Do not mark a task as "done" before updating the docs
- Describe what sections are changed in pull requests
- Before approving pull requests, ensure docs are updated
- Do regular code reviews where you check for updated docs





Reference(s): M, Novikova, Savonen, Veerman

Creating Checklists

- ACCEU can be used to guide the creation of checklists for documenting a program file
- But a checklist is not the be-all, end-all; be flexible and use discretion
- Different programs require different details to be documented