5. Java

Characteristics of Java

- OOP-focused language – rejects procedural programming
- Enforced error catching
- Automatic garbage collection
- Javadoc documentation
- JUnit testing
- Android programming

Common pitfalls of Java

- Java cares about how code is stored in your filesystem
  - Each public class is one file
  - Each package is one folder
- Java is heavily OOP
  - All code must be in a function
  - All functions must belong to a class
  - Get used to inheritance, encapsulation, constructors, etc.
- Questions about speed, performance

Hello World

```
class HelloWorld {
   public static void main (String[] args) {
      System.out.println(“Hello World”);
   }
}
```

- Needs to be in a file called HelloWorld.java, compiled with javac HelloWorld.java, executed with java HelloWorld
- main is a special function that indicates start of code execution
- HelloWorld is a class without a constructor
- return type void – no return

Access modifiers

- Public – accessible to anything
- Private – accessible to no other class
  - Even subclasses
- Protected – accessible to packages and subclasses
- Default (“Package private”) – accessible to packages, not subclasses

Class constructor

```
class Dog {
   String mood = “sad”;
   public Dog() {
      this.mood = “happy”;
   }
   public setMood(String moodString) {
      mood = moodString;
   }
}
```

- What is the value of:
  - Dog.mood?
  - Dog myDog = new Dog(); myDog.mood?
  - Dog myDog = new Dog(); myDog.setMood(“cheerful”); myDog.mood?

Wrong use of Static

```
class Dog {
   static String mood = “sad”;
   public Dog() {}
   public static setMood(String moodString) {
      mood = moodString;
   }
}
```

1. Dog myDog1 = new Dog();
2. Dog myDog2 = new Dog();
3. myDog1.setMood(“cheerful”);

What is myDog2.mood?

Proper use of Static

```
class Dog {
   final static scienceName = “canine”;
   static int dogCount = 0;
   public Dog() {dogCount += 1;}
}
```

- Static is a member of the class itself, not any of its instances
  - It can also be thought of as being shared by all its instances
- A non-final static is useful for tracking “global” state of a class
- Fully static classes are utility classes
- static final variable is a constant

Wrong use of Inheritance

```
class Animal {
   String animalType;
   public Animal () {//…}
   public void moveCycle() {
      if (animalType.equals(“dog”)) {
         //dog behavior…
      }
      if (animalType.equals(“cat”)) {
         //cat behavior…
      }
   }
}
```

```
class Dog extends Animal {
   public Dog () {
      animalType = “dog”;}
   public moveCycle() { }
}
```

- Dog myDog = new Dog();
- myDog.moveCycle();

- This is wrong in two ways

Proper use of Inheritance

```
class Animal {
   public Animal () {//…}
   public void moveCycle() {//
      //default behavior…
   }
}
```

```
class Dog extends Animal {
   public Dog () {//…}
   public moveCycle() {
      //dog behavior…
   }
}
```

- Inherited subclass should extend/override functionality of parent class
- Abstract method must be in abstract base class, cannot be declared
- Cannot multiple inherit in Java

Proper use of Inheritance

- Some rules to consider:
    - Composition over Inheritance?
    - Parents should never have code for subclasses
    - Do not inherit if you would discard functionality
    - Implementation classes vs Domain classes

```
class CustomerGroup extends ArrayList<Customer> {
   //…
}
```

```
class CustomerGroup{
   ArrayList<Customer> listCustomers;
   //…
}
```

Interface Implementation

- Interface is similar in usage to abstract base class
  - What are the differences?

```
public interface Animal {
   public void moveCycle();
}
```

```
public interface GUIObject {
   public void paint();
}
```

```
class Dog implements Animal, GUIObject {
   public void moveCycle() {//…}
   public void paint() {//…}
}
```

Generic types (templates)

- A class and function can be declared for generic type
- Can also bound method with extends

```
class numDictionary<N extends Number, V> {
    private N key; private V value;
    public numDictionary() {//…}
    public boolean isGreater(inNum) {
        return (inNum > key);
    }
    public void printValue() {
        System.out.println(value.toString());
    }
    public V getValue() {
        return value;
    }
}
```

Anonymous classes

- Sometimes you want to declare and use a class at the same time
- Type of inner class
- Example: JPanel.addMouseListener takes a MouseListener object, which MouseAdapter implements

```
public class GamePanel extends JPanel {
   addMouseListener(new MouseAdapter() {
      public void mouseEntered(MouseEvent e) {
         System.out.println(“Hello little mouse!”);
      }
   }
}
```

Lambda expressions

- Simplifies function calls
- Simplifies interface implementation

```
for (Rat rat: rats) {
   rat.check_caught();
}
```

```
rats.forEach(rat ->
{rat.check_caught();})
```

```
interface Animal {
   public String makeSound();
}
```

```
Dog dog = ()-> {return “woof”;}
System.out.println(dog.makeSound());
```

More lambda expressions

```
interface Predicate<Person> {
    boolean test(Person t);
}
```

```
public static void printPersonsPredicate(
    List<Person> roster, Predicate<Person> tester)
{
    for (Person p : roster) {
        if (tester.test(p)) {
            p.printPerson();
        }
    }
}
```

```
printPersonsPredicate(
    roster,
    p -> p.getGender() == Person.Sex.MALE
        && p.getAge() >= 18
        && p.getAge() <= 25
);
```

Principles of OOP

- How are these principles/properties implemented in Java?
  - Separation of concerns
  - Open/closed principle
  - Liskov substitution principle
  - Polymorphism: a piece of code can handle different types
- How do you program well to achieve these goals?
  - Maximize cohesion
  - Minimize coupling

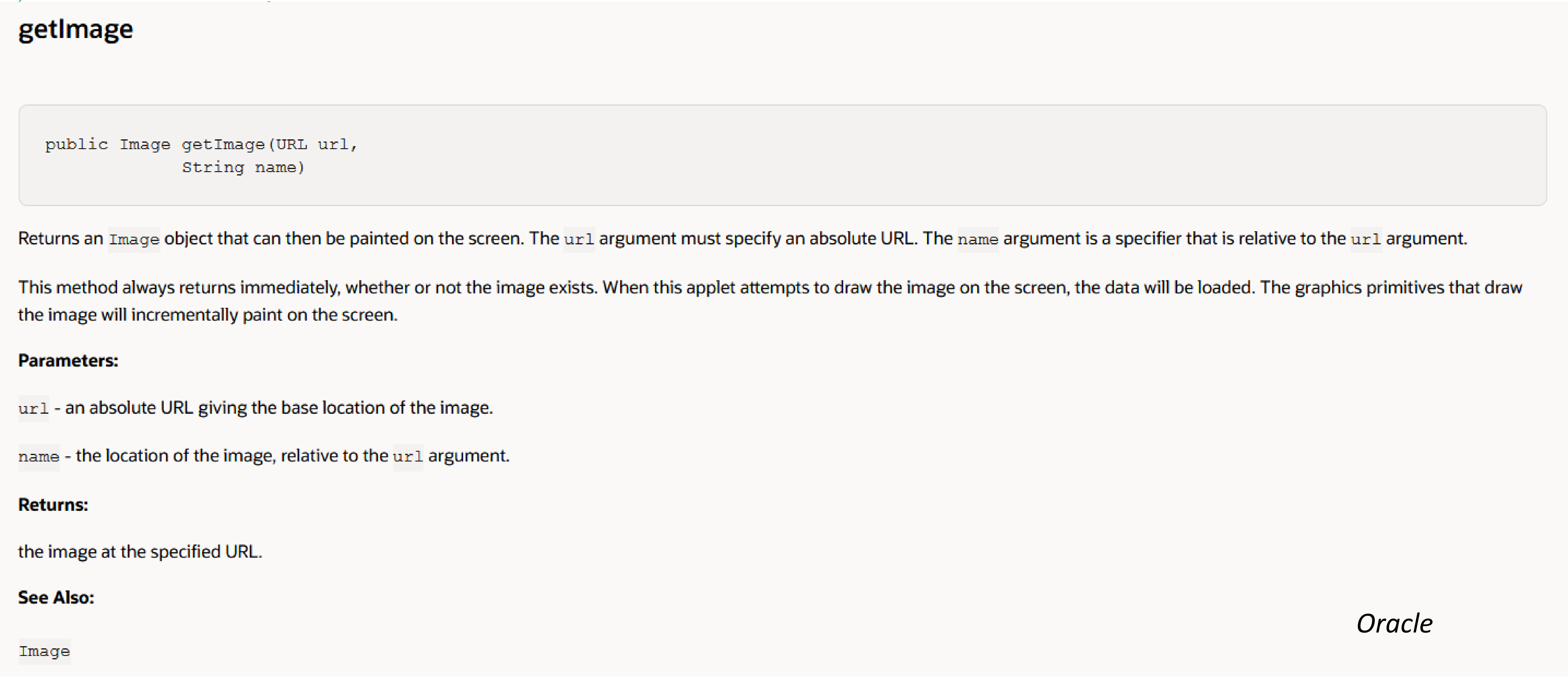
Javadoc

- Specific documentation format that allows automatic generation of documentation
    - Document is generated as HTML
- Helps document inheritance/implementation
- Rules:
    - Starts with /** in the first line, ends with */ in the last
    - Put before each protected/public function, fields
    - <80 characters per line, <p> for new paragraph
    - Define input params with @param, return values with @return
    - First sentence is always a short summary (ends with .)

Javadoc

- /**
- * Returns an Image object that can then be painted on the screen.
- * The url argument must specify an absolute <a href="#{@link}">{@link URL}</a>. The name
- * argument is a specifier that is relative to the url argument.
- * <p>
- * This method always returns immediately, whether or not the
- * image exists. When this applet attempts to draw the image on
- * the screen, the data will be loaded. The graphics primitives
- * that draw the image will incrementally paint on the screen.
- *
- * @param url an absolute URL giving the base location of the image
- * @param name the location of the image, relative to the url argument
- * @return   the image at the specified URL
- * @see   Image
- */

Oracle



getImage

```
public Image getImage(URL url,
                      String name)
```

Returns an `Image` object that can then be painted on the screen. The `url` argument must specify an absolute URL. The `name` argument is a specifier that is relative to the `url` argument.

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

`url` - an absolute URL giving the base location of the image.

`name` - the location of the image, relative to the `url` argument.

Returns:

the image at the specified URL.

See Also:

`Image`

Oracle

Javadoc

- Javadoc should be an API guide, not a programming guide
  - Avoid implementation details or bugs/workarounds
    - Use a custom tag such as @bug for such
  - Do describe dependencies
- For overloaded functions, summary sentences should be different for each function
- Prefer to be clear over concise
- There are many other rules/guidelines/features; Oracle has a Javadoc tutorial with a style guide