U N I V E R S I T Y

CS 180: Problem Solving and Object-Oriented Programming

Lecture 10: Methods and Classes

Dr. Jeff Turkstra

Lecture 10

- Constructors
- Scope and Extent
- Overloading
- Initialization blocks
- Variable arguments
- Packages and JAR files

Attacking a problem

- Top down (understanding the problem)
  - What should the program do?
  - What are its "big steps"?
  - How would you solve it without a program?
- Bottom up (building and testing)
  - What "little problems" can you solve and test?
  - What Java classes and objects can you use?
  - What classes and methods do you need to build?

Working as a team

- Collaboration requires dividing up the work
- In Java, this division is often on class boundaries
- Alice writes class ImageManipulation
- Bob writes class ReadWriteImages
- Alice doesn't worry about how Bob reads and writes the images
- Bob doesn't worry about how Alice does the image manipulation

Pass by value

- Java passes arguments by value
- Even for objects
  - But, it's the reference that is passed by value
  - The object itself is not copied

Constructors

- Create objects from the class "blueprint"
- Similar to method declarations
  - Use class name
  - No return type
- Provided automatically if not specified
  - Be careful

Constructors

- Similar to a method
  - Not directly invoked
- Invoked by new
  - Or another constructor
- May access all class fields
- Does not explicitly return a value
- But, the new operator returns a reference

Scope

- Where in the code a variable is usable
- Basic rule: a variable is usable from the point of declaration to the end of the enclosing block
- Cannot have two variables with the same name active in same scope
  - Except: parameters and local variables can shadow fields with the same name

this Keyword

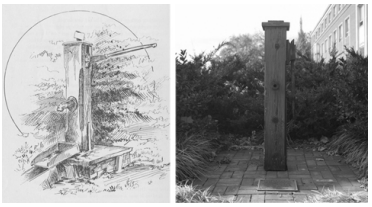
- Most commonly used if a field is shadowed by a method or constructor parameter
  - Look at wheel again
- Can also be used to call another constructor in the same class
  - Explicit constructor invocation
- Reference to the “current object”

Extent

- How long a value is maintained
  - Its “lifetime”
- Local variables: same as scope
- Parameters: created on method call, destroyed on return
- Objects: created by new, destroyed when garbage collected
  - Reference counting

Purdue Trivia

- The Old Pump sits southeast of Stone Hall, commemorating a tradition of friendship and romance
- Also central to pranks by the “dorm devils,” an early 19th-century group of students living in Purdue Hall
  - They even got President Smart
- Now has a plaque: “Older than Purdue itself, this pump was once a meeting place for campus sweethearts. Today it symbolizes romance and friendship and the spirit of Purdue. In recognition of the contribution of the Purdue woman to the preservation of this spirit, this pump is dedicated. Purdue Reamer Club, 1958.”



Overloading

- Method signature: Method name & parameter types
- Signatures must be unique
  - Only way the compiler can tell the difference between methods
- Method names do not have to be unique
- Overloading: same method name with a different parameter list

- print(), println(), etc
  - https://docs.oracle.com/javase/10/docs/api/java/io/PrintStream.html

Improving isPalindrome

```
public class Palindrome {
  static boolean isPalindrome(String s) {
    if (s == null)
      return true;

    for (int i = 0; i < s.length() / 2; i++)
      if (s.charAt(i) != s.charAt(s.length() - 1 - i))
        return false;
    return true;
  }

  static boolean isPalindrome(int x) {
    return isPalindrome(Integer.toString(x));
  }
}
```


this again

- this can be used to invoke one constructor from another
  - Must be the first statement
- Still relies on the signature
- Returns to calling constructor when done


```
public class PurdueStudent {
  boolean hasName;
  String name;
  int puid;

  // constructor with int...
  PurdueStudent(int puid) {
    this.puid = puid;
    hasName = false;
  }

  // constructor with String and int...
  PurdueStudent(String name, int puid) {
    this(puid);
    this.name = name;
    hasName = true;
  }

...
```



```
  void printStudent() {
    if (hasName)
      System.out.println(name + ": " + puid);
    else
      System.out.println("(no name): " + puid);
  }

  public static void main(String[] args) {
    // call int-only constructor...
    PurdueStudent p1 = new PurdueStudent(1010337138);

    // call String-int constructor...
    PurdueStudent p2 = new PurdueStudent("Drake", 1123441245);

    p1.printStudent();
    p2.printStudent();
  }
}
```

Initialization blocks

- Instance
  - Executed on object creation, before the constructor
  - After static and super class initialization blocks/constructors
- Static
  - Executed once at class load time

Variable arguments

- Called varargs
- Allows an arbitrary number of values to be passed to a method
  - Remember printf?
- Shortcut to creating an array manually
  `someMethod(type… name)`

Packages

- Are created using the package keyword. E.g.,
  `package letters;`
- This also specifies a directory hierarchy
- Not using a package statement causes your class(es) to end up in an unnamed package
  - Fine for small or temporary applications

JAR files

- Mechanism for bundling multiple files into a single archive
  - Literally uses the zip algorithm
- Why?
  - Security – digital signatures
  - Decreased download time (compression)
  - Versioning
  - Portability

JAR

- Create
  $ jar cf jar-file input file(s)
- View
  $ jar tf jar-file.jar
- Extract
  $ jar xf jar-file.jar
- Update
  $ jar uf jar-file input file(s)

- Can package entire programs
  - Must specify Main-Class in the manifest

Boiler Up!