Java - What is it?

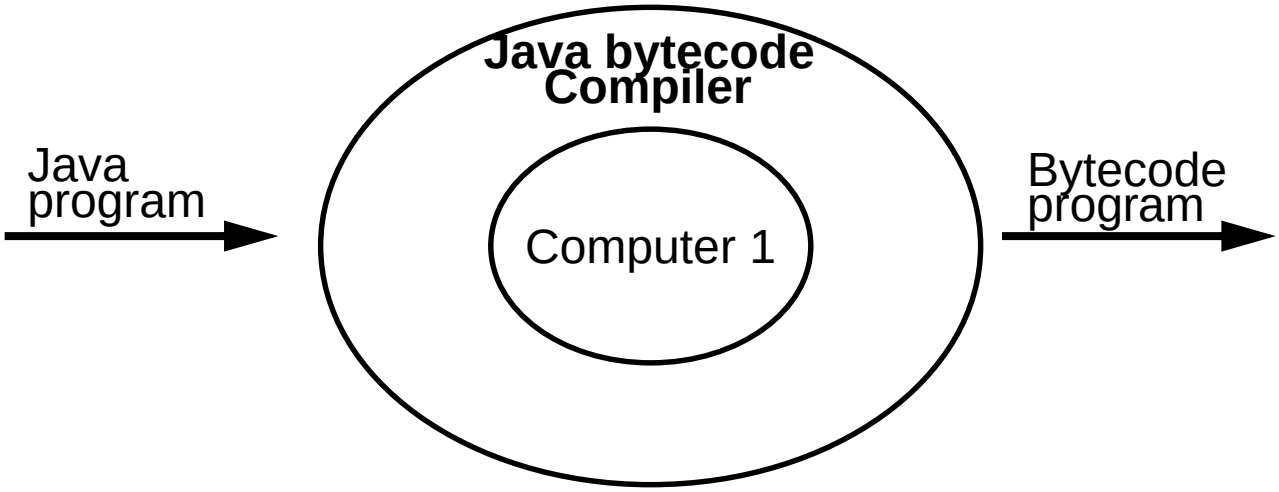
.. a portable, object-oriented, "network-savvy" language supporting:

- **inheritance, abstract classes, and interfaces**
  - "Program to an interface, not an implementation"
    - *Design Patterns*, Gamma et al.
- **polymorphism**
  - same name, many forms..
- **event driven programming**
  - listen and observe..
- **threads**
  - separate processes..
- **automatic garbage collection**
  - no memory management!
- **graphics libraries**
  - AWT .. on Thursday

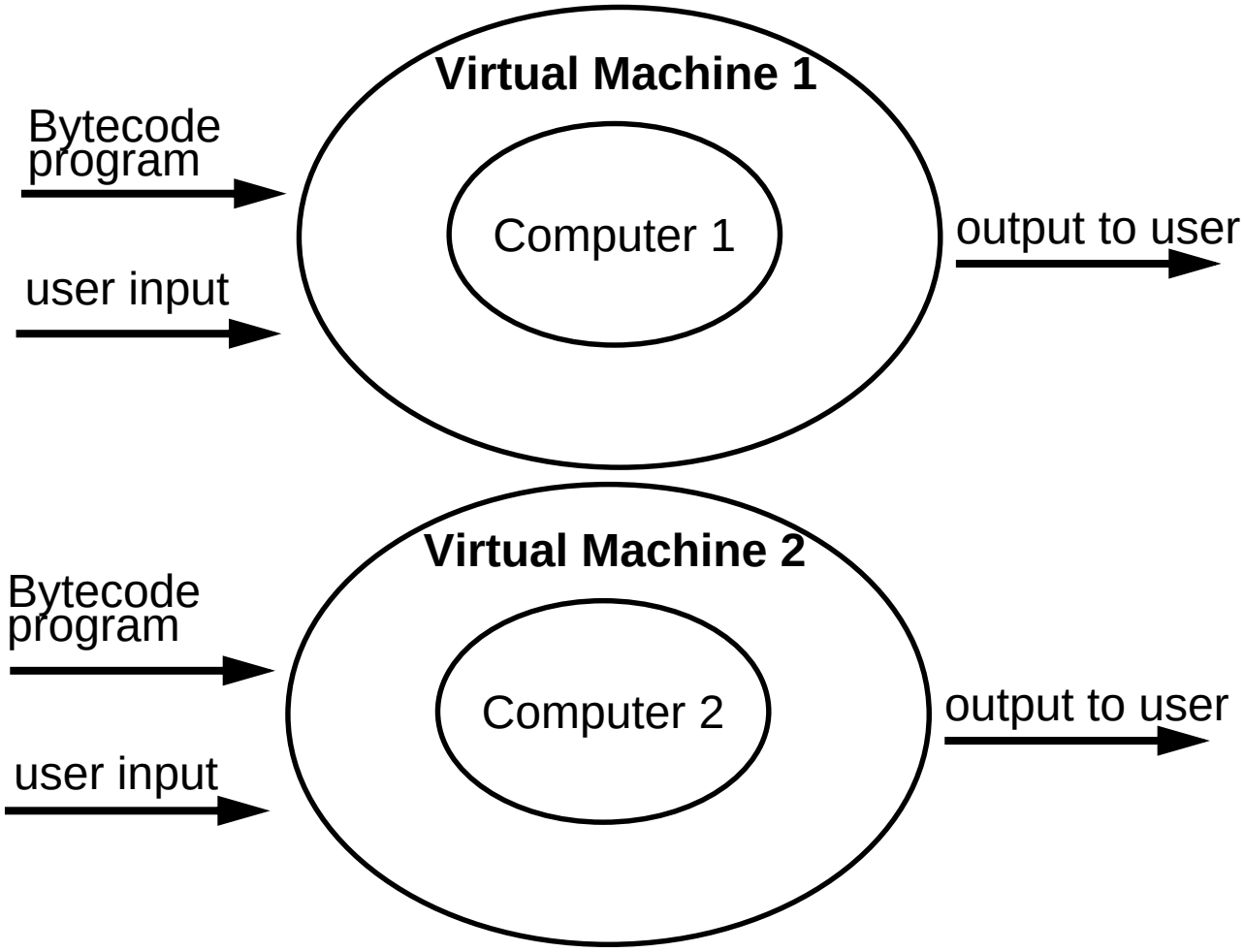
These features can improve development time and code readability, but they require practice!

Java is portable ..

- Compilation



- Execution (by interpretation)



Java is object oriented ..

```
public class HelloWorldProgram {
   public static void main(String argv[]) {
      System.out.println("Hello!");
   }
}
```

- Type the program into a file with the class name:

  HelloWorldProgram.java

- Compile into bytecode with:

  javac HelloWorldProgram.java

- This produces a class file:

  HelloWorldProgram.class

- Run the program:

  java HelloWorldProgram

Java is object-oriented ..

Encapsulation: objects have a public interface

```
public class ComplexNumber {
    // public interface ...
    public double getRealPart() {}
    public void setRealPart(double real) {}

    public double getImagPart() {}
    public void setImagPart(double imag) {}

    // implementation details...
    ...
}
```

Use getters and setters to hide implementation details!

Java is object-oriented ..

Encapsulation: protecting implementation details

- **public** means accessible to the world:

```
public static final double PI = 3.1415927;
```

- **private** means accessible to instances of this class only:

```
private String text_ = "Hello World!";
```

- **protected** means accessible to the package and to subclasses:

```
protected boolean  status_ ;
```

- no protection keyword means accessible to the package:

```
String packageName_;
```

Java is “network-savvy”..

Creating an Applet

```
import java.applet.Applet;
import java.awt.*;

public class HelloWorldApplet extends Applet {
   private static final String text_ = "Hi!";

   // called on start of applet
   public void init() {}

   // called to refresh appearance
    public void paint(Graphics g) {
      Dimension d = getSize();
      FontMetrics f = g.getFontMetrics();
      int sWidth = f.stringWidth(text_);
      int sHeight = f.getHeight();
      int xOrigin = (d.width - sWidth) / 2;
      int yOrigin = (d.height + sHeight) / 2;
      g.drawString(text_, xOrigin, yOrigin);
   }
}
```

Java is “network-savvy”..

Creating a web page

- Type program into a file with the class name:

```
HelloWorldApplet.java
```

- Compile into bytecode with:

```
javac HelloWorldApplet.java
```

- This produces the class file:

```
HelloWorldApplet.class
```

- Make test.html file with applet tag

```
<html>
<head><title>Hello World</title></head>
<body>
<applet code="HelloWorldApplet.class"
        width=200 height=200></applet>
</body>
</html>
```

- Load web page into browser, or use:

```
appletviewer test.html
```

Java is “network-savvy”..

Security issues

- applets cannot run a local executable program
- applets cannot read from or write to the local file system
- applets can only communicate with the server from which they originate
- applets can only learn a few facts about the local computer (e.g. operating system)

Question

```
public class HelloWorldApplet extends Applet {
   private static final String text_ = "Hi!";
    ...
   public void paint(Graphics g) {
      ...
     int xOrigin = (d.width - sWidth) / 2;
     int yOrigin = (d.height + sHeight) / 2;
     g.drawString(text_, xOrigin, yOrigin);
   }
}
```

What actually gets copied and passed to g.drawString?

Inheritance and Abstract Classes

```
public abstract class Shape {
   public abstract double area();
   ...
   public void move () {...}
}

public class Rectangle extends Shape {
   public double area() {
         return width_ * height_;
    }
    // private instance variables..
    private double width_;
    private double height_;
}
```

- Create a subclass --
  - **To specialize:** a rectangle **“is a”** shape that **“has a”** width and height
  - **To specify:** the rectangle class specifies how area is computed
  - **For code reuse:** rectangle can use Shapes move method

Question

- Java allows single inheritance only.
- What if we want to combine sets of properties?
- Example:
  - ListeningButton **“is-a”** Button
  - ListeningButton **“is-a”** ActionListener

Question: What if we want to combine sets of properties?

Answer: Interfaces!

```
import java.awt.event.*;

interface ActionListener {
  public void actionPerformed (ActionEvent e);
}

class ListeningButton extends Button implements
ActionListener {
  public ListeningButton () {
    addActionListener (this);
  }
  public void actionPerformed (ActionEvent e) {
    // do the button action..
  }
}
```

- Create an interface --
  - To impose requirements outside the inheritance heirarchy
  - As a type indicator

Polymorphism - same name, many forms

```
public class Rectangle extends Shape {
  // parametric overloading
  public Rectangle () {}
  public Rectangle (int width, int height) {}
  public Rectangle (int width, int height,
                    int xLoc, int yLoc) {}
  public Rectangle (Rectangle anotherRect) {}
  // override Shape area method
  public double area () {}
}

Shape s = new Rectangle();
Object o = s;
```

- Three kinds of polymorphism:
  - **Parametric overloading:** A function name can denote different functions
  - **Overriding:** A method can override a method of the same name in a parent class
  - **Polymorphic variables:** A variable can hold different types of values
- When are polymorphic variables useful?

Event Driven Programming

Problem: How do you capture external (user) events?

Event Driven Programming

Listeners

Problem: How do you capture external (user) events?
Solution: Register listeners!

```
import java.awt.event.*;

interface ActionListener {
  public void actionPerformed (ActionEvent e);
}

class ListeningButton extends Button implements
ActionListener {
  public ListeningButton () {
    // we are our own listener..
    addActionListener (this);
  }
  public void actionPerformed (ActionEvent e) {
    // do the button action..
  }
}
```

Event Driven Programming

Listeners

- User interface components maintain collections of listener objects.
- When a component changes state, it notifies all of its listeners.
- The collection of listener objects is dynamic.. listeners can be added and removed at run time.
- More details in later classes .. (also see cs4 Java notes)

Event Driven Programming

Problem: How to synchronize the behavior of two separate, but loosely connected objects?

Example: An app that tracks user logins.

Observables and Observers

Problem: How to synchronize the behavior of two separate, but loosely connected objects?

One Solution: Create Observables and Observers

```
import java.util.*;
public class LoginServer extends Observable {
  public void login(User newUser) {
    setChanged();
    notifyObservers(newUser);
  }
}

public class BigBrother implements Observer {
  public LoginServer server_;
  public BigBrother () {
    server_ = new LoginServer();
    server_.addObserver(this);
  }
  public void update (Observable o, Object x) {
    User newUser = (User) x;
    System.out.println("User "+ newUser.name +
                       "logging in");
  }
}
```

Threads

Problem: an application may have multiple logically separate processes

One Solution: threads!

```
public class DirectDepositor extends Thread {
  private double amount_;
  private BankAccount account_;
  public void run() {
    while (true) {
      account.deposit(amount_);
      try{sleep(Time.oneMonth);}
      catch (InterruptedException e) {return;}
    }
  }
}
public class BankApp extends Applet {
  private BankAccount account_;
  public void init() {
    account_ = new BankAccount();
    DirectDepositor depositServer =
        new DirectDepositor(account_);
    depositServer.start();
  }
}
```

Threads

Problem: what if two threads try to deposit at once?

```
public class BankAccount {
    private double balance_;
    ...
    public void deposit(double amount) {
        balance_ += amount;
    }
}
```

Threads

Problem: what if two threads try to deposit at once?
Solution: synchronized

```
public class BankAccount {
  private double balance_;
  ...
  public synchronized void deposit(double d) {
    balance_ += d;
  }
}
```

Java - What is it?

.. a portable, object-oriented, "network-savvy" language supporting:

- **inheritance, abstract classes, and interfaces**
  - "Program to an interface, not an implementation"
    - *Design Patterns*, Gamma et al.
- **polymorphism**
  - same name, many forms..
- **event driven programming**
  - listen and observe..
- **threads**
  - separate processes..
- **automatic garbage collection**
  - no memory management!
- **graphics libraries**
  - AWT .. on Thursday

These features can improve development time and code readability, but they require practice!