Outline

■ References
■ Methods
■ Messages and methods
■ Reference Variables
■ Order of statements
References

- Reference – phrase that refers to object
- Ex. System.out
- Reference is not object, but usually say
  - “the System.out object”, instead of
  - “the object referred to by System.out”
- Can send message to the referred to object
- Can also send object reference as part of message
Methods

■ Method
  - Section of Java code that provides a behavior
  - Has a method name and arguments
  - Arguments are additional information sent in message

■ Message has form

  *method-name* ( *arguments*)
Example Methods

- PrintStream printing methods
  - println with string to print
  - println with no string
  - print with string to print

- Each provides different behavior
Different methods provide different behavior

System.out.println(“Hokie High!”);  →  Hokie High!

System.out.println(“Hokie”);
System.out.println(“High!”);

System.out.print(“Hokie ”);
System.out.print(“High!”);
System.out.println();
More References

- String class – represents strings of characters: names, addresses, etc.
- String literal constant – reference to an object
  - Ex: “Hokie High!”
  - Literal because it literally describes object
  - Constant because we can’t change it
- String class also has methods used to send messages to string objects
- Remember form of statement to send message
  \[ \text{reference} . \text{method-name} ( \text{arguments} ) \]
- So can write
  \[ "vpi".toUpperCase() \]
- Recipient is object referred to as “vpi”
- Says: give a string that is you in upper case
- Sending code execution temporarily suspended until receiving object receives the message, handles it and completes execution.
String toUpperCase Method

- The toUpperCase does not modify object
- Creates new object that is uppercase version
- So it is legitimate to write
  
```java
  System.out.print("vpi".toUpperCase());
  ```
3. Using Objects

References in Method Call

Sending message to object

PrintStream object reference

String object reference

Sending object as argument

String object reference

System.out.print(“vpi”.toUpperCase());
More on Methods

- Describe methods using
  - *signature* – the method name and its arguments
  - *prototype* – signature plus return value
- Following slide shows prototypes for methods we’ve seen so far
## Method Inventory

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Returns</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrintStream</td>
<td>println</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>PrintStream</td>
<td>println</td>
<td>none</td>
<td>reference to a String object</td>
</tr>
<tr>
<td>PrintStream</td>
<td>print</td>
<td>none</td>
<td>reference to a String object</td>
</tr>
<tr>
<td>String</td>
<td>toUpperCase</td>
<td>reference to a String object</td>
<td>none</td>
</tr>
</tbody>
</table>
Overloaded Methods

- Method name is *overloaded* if there are at least two methods with the same name but different arguments – different signature

- Examples:
  - `println` in `PrintStream` class
  - `substring` in `String` class (p.38 of text)

- Overloading allows objects to provide similar behavior that depends on arguments
Immutable Objects

- String class provides no behavior that allows an object to be modified
- All methods return a reference to a new string
toUpperCase is an example
- Strings are *immutable*
- Objects that can be modified are *mutable*
Reference Variables

- **Variable** – identifier that can be given a value
- **Reference variable** – value is a reference
- Allows saving a reference for later use
- Can keep us from duplicating computations
- Ex: typing
  
  ```java
  System.out.print(cheer);
  ```

  instead of
  
  ```java
  System.out.print(“Hokie, Hokie, Hokie Hi!”);
  ```

cheer

“Hokie, Hokie, Hokie Hi!”
Declaring Reference Variables

- Declaration is another kind of statement
- Example:
  
  ```java
  String cheer; // reference variable for string
  ```
- Can also initialize with value
  
  ```java
  String cheer = “Hokie, Hokie, Hokie, Hi!”;
  ```
- General form of declaration
  
  ```java
  type identifier, identifier, …;
  ```
  where, for now, type is a class name

```java
string moonDate,              // date of 1st moon landing
leniodesDate,                // meteor shower date
vEquinox;                    // Vernal Equinox
```
Saving References

- Assignment statements
  ```java
  cheer = "Hurrah!";
  ```
- Assigns to cheer the reference to the object “Hurrah!”
- Saves reference for later use
  ```java
  System.out.print(cheer);
  System.out.print(cheer);
  ```
- Replaces reference to any object previously held

```
cheer

"Hokie, Hokie, Hokie Hi!"
```

```
"Hurrah!"
```
Retrieving References

- Have seen that reference variable can be used when passing message
  ```java
  System.out.print(cheer);
  ```
- May also be used in assignment statements
  ```java
  String startMonth, current;
  startMonth = "August";
  current = startMonth;
  ```
- Distinguish between left-hand and right-hand sides of assignment

![Diagram of assigning "August" to startMonth and current]
Aliases

- Two reference variables to same object are called aliases
  
  ```java
  String startMonth, current;
  startMonth = "August";
  current = startMonth;
  ```

- Assigning another reference to a variable does not affect its aliases
  
  ```java
  current = "September";
  ```

```
startMonth

```

```
current

```
Type and Assignment

- The type of a variable determines what can be assigned to it
  
  ```java
  String s = “Hello”;
  PrintStream p = System.out;
  s = “Sam”; // correct type
  s = p; // wrong type!
  ```

- A PrintStream object is not a String object
Assignment is not Equality

Even though equal sign is used assignment does not represent equality

Example:

```java
current = "September";
```

This does not mean that

- Prior to statement `current` holds reference to this string
- `current` will always refer to this string

Is an *imperative* – “do this…”
Statement Inventory

- Declarations – what variables, what type of object they can refer to
  
  ```java
  String startMonth, current;
  ```

- Assignment statements – saving reference values
  
  ```java
  startMonth = “August”;
  current = startMonth;
  ```

- Message-passing statements
  
  ```java
  current = “September”.toUpperCase();
  ```
Order of Statements

- Java statements are executed from first to last
  - Sequential/Default execution

- Some statements must occur in a particular order
  - Declaration of a variable before it is assigned to
    ```
    String s;
    s = “first and ten”;
    ```
  - Assignment/initialization of a variable before used as message argument or in
    right-hand side of assignment
    ```
    cheer = “Rah!”;
    s = cheer.toUpperCase();
    System.out.print(s);
    ```
Equivalent Statement Sequences

String cheer;
String chant;
cheer = “Rah!”;
chant = “Please give us the grade we want”;