Binary
**Binary**

**Bits**

- *Binary digits*: 0 or 1; base 2 numbers.
- Information in computers is encoded in binary.
**Binary**

**Bits**

- *Binary digits*: 0 or 1; base 2 numbers.
- Information in computers is encoded in binary.

**Decimal ↔ Binary**

198 =
Binary

Bits

- *Binary digits*: 0 or 1; base 2 numbers.
- Information in computers is encoded in binary.

Decimal ↔ Binary

\[198 = 1 \cdot 10^2 + 9 \cdot 10^1 + 8 \cdot 10^0\]
**Binary**

**Bits**
- *Binary digits*: 0 or 1; base 2 numbers.
- Information in computers is encoded in binary.

**Decimal ↔ Binary**

\[
198 = 1 \cdot 10^2 + 9 \cdot 10^1 + 8 \cdot 10^0 \\
= 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0
\]
Binary

Bits

- Binary digits: 0 or 1; base 2 numbers.
- Information in computers is encoded in binary.

Decimal ↔ Binary

\[
198 = 1 \cdot 10^2 + 9 \cdot 10^1 + 8 \cdot 10^0 \\
= 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\
= 11000110_2
\]
TopHat Question 1

What is 17 in binary?

a. 10000
b. 10001
c. 11001
d. 10011
TopHat Question 2

What is 10110 in decimal?

a. 18
b. 20
c. 22
d. 24
Numbers in Bases of Powers of 2

Decimal Base 10 \( \approx 2^{3.32193} \): \( 21_{10} \) or 21.
Numbers in Bases of Powers of 2

Decimal  Base $10 \approx 2^{3.32193}$: $21_{10}$ or 21.

Unary  Base $2^0 = 1$: 1111111111111111111.
Numbers in Bases of Powers of 2

**Decimal**  Base 10 ≈ $2^{3.32193}$: 21_{10} or 21.

**Unary**  Base $2^0 = 1$: 11111111111111111_{1}.

**Binary**  Base $2^1 = 2$: 10101_2 or 0b10101 (or 10101 when the base is clear from the context).
Numbers in Bases of Powers of 2

Decimal  Base 10 ≈ 2^{3.32193}: 21_{10} or 21.

Unary   Base 2^0 = 1: 1111111111111111_{1}.

Binary  Base 2^1 = 2: 10101_{2} or 0b10101 (or 10101 when the base is clear from the context).

Quarternary  Base 2^2 = 4: 111_{4}.
Numbers in Bases of Powers of 2

**Decimal** Base $10 \approx 2^{3.32193}$: $21_{10}$ or 21.

**Unary** Base $2^0 = 1$: 111111111111111111111111.

**Binary** Base $2^1 = 2$: 10101₂ or 0b10101 (or 10101 when the base is clear from the context).

**Quarternary** Base $2^2 = 4$: 11₁₄.

**Octal** Base $2^3 = 8$: 25₈ or 025 or 0o25.
**Numbers in Bases of Powers of 2**

**Decimal** Base $10 \approx 2^{3.32193}$: $21_{10}$ or 21.

**Unary** Base $2^0 = 1$: $111111111111111111_1$.

**Binary** Base $2^1 = 2$: $10101_2$ or $0b10101$ (or $10101$ when the base is clear from the context).

**Quarternary** Base $2^2 = 4$: $111_4$.

**Octal** Base $2^3 = 8$: $25_8$ or $025$ or $0o25$.

**Hexadecimal** Base $2^4 = 16$: $15_{16}$ or $0x15$.

Need 16 characters to represent the 16 digits:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 → 0</td>
<td>0</td>
</tr>
<tr>
<td>1 → 1</td>
<td>1</td>
</tr>
<tr>
<td>2 → 2</td>
<td>2</td>
</tr>
<tr>
<td>3 → 3</td>
<td>3</td>
</tr>
<tr>
<td>4 → 4</td>
<td>4</td>
</tr>
<tr>
<td>5 → 5</td>
<td>5</td>
</tr>
<tr>
<td>6 → 6</td>
<td>6</td>
</tr>
<tr>
<td>7 → 7</td>
<td>7</td>
</tr>
<tr>
<td>8 → 8</td>
<td>8</td>
</tr>
<tr>
<td>9 → 9</td>
<td>9</td>
</tr>
<tr>
<td>10 → a</td>
<td>a</td>
</tr>
<tr>
<td>11 → b</td>
<td>b</td>
</tr>
<tr>
<td>12 → c</td>
<td>c</td>
</tr>
<tr>
<td>13 → d</td>
<td>d</td>
</tr>
<tr>
<td>14 → e</td>
<td>e</td>
</tr>
<tr>
<td>15 → f</td>
<td>f</td>
</tr>
</tbody>
</table>
TopHat Question 3

Convert 1100 1010 1111 1110 1011 1010 1011 1110 to hexadecimal.

Type your answer in TopHat.
Reference Types
REFERENCES FOR PRIMITIVES

<table>
<thead>
<tr>
<th>Reference</th>
<th>Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>int</td>
</tr>
<tr>
<td>Long</td>
<td>long</td>
</tr>
<tr>
<td>Double</td>
<td>double</td>
</tr>
</tbody>
</table>

Primitive Wrappers

- Objects that contain a primitive value.
- Allows primitives to be used in many of Java’s built-in data structures like ArrayList.
- Provides some addition methods for conversions.
# References for Primitives

<table>
<thead>
<tr>
<th>Reference</th>
<th>Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>int</td>
</tr>
<tr>
<td>Long</td>
<td>long</td>
</tr>
<tr>
<td>Double</td>
<td>double</td>
</tr>
</tbody>
</table>

E.g.

```java
Integer intWrap = 5;
Long longWrap = new Long(15);
```

## Primitive Wrappers

- Objects that contain a primitive value.
- Allows primitives to be used in many of Java’s built-in data structures like `ArrayList`.
- Provides some addition methods for conversions.
References for Primitives

<table>
<thead>
<tr>
<th>Reference</th>
<th>Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>int</td>
</tr>
<tr>
<td>Long</td>
<td>long</td>
</tr>
<tr>
<td>Double</td>
<td>double</td>
</tr>
</tbody>
</table>

E.g.

Integer intWrap = 5;
Long longWrap = new Long(15);

Primitive Wrappers

- Objects that contain a primitive value.
- Allows primitives to be used in many of Java’s built-in data structures like ArrayList.
- Provides some addition methods for conversions.
- `typeValue()` method for retrieving primitive. E.g. `intWrap.intValue();`
What are References?

Less Simple Memory Model
What are References?

Less Simple Memory Model

```java
int i = 5;
```

Stack

Heap
**What are References?**

**Less Simple Memory Model**

```java
int i = 5;
Integer j = 6;
```
What are References?

Less Simple Memory Model

```
int i = 5;
Integer j = 6;
Double d = 2.5;
```
What are References?

Less Simple Memory Model

Stack

- `int i = 5;`
- `Integer j = 6;`
- `Double d = 2.5;`
- `Integer s = 6;`

Heap

- 6
- 2.5
What are References?

Less Simple Memory Model

```java
int i = 5;
Integer j = 6;
Double d = 2.5;
Integer s = 6;
Scanner sc =
    new Scanner(System.in);
```
**What are References?**

**Less Simple Memory Model**

```
int i = 5;
Integer j = 6;
Double d = 2.5;
Integer s = 6;
Scanner sc =
    new Scanner(System.in);
```

**References**
- Refer to an object.
- Value: referral information.
Strings
Characters

Primitive

- `char c;`
- Literal: ’a’
  Note: single quotes (’) not double quotes (").
- Wrapper Object: Character
Characters

Primitive
- char c;
- Literal: ’a’
  Note: single quotes (’) not double quotes (").
- Wrapper Object: Character

Escape Characters
- \: Escape metacharacter.
  \n – Newline   \’ – Single quote   \t – Tab
  \r – Carriage return   \" – Double quote   \b – Backspace
  \f – Line feed   \\ – Backslash
Character Encodings

ASCII

- A 7-bit character encoding in 1 byte.

UNICODE

- An ASCII extension using more bits.
- UTF-8: 1 to 4 bytes; most popular UNICODE. More popular than ASCII as encoding on the web since 2008. Currently, about 90% of the web.a
- UTF-16: 2 or 4 bytes; used by java internally to encode strings.

---

ahttps://w3techs.com/technologies/overview/character_encoding/all
**Strings**

Declaration:

```java
String aStr = "hello world!";
```

- **String**
  - A reference type.
  - Refers to a sequence of characters.
Strings

Declaration:

String aStr = "hello world!";

String

- A reference type.
- Refers to a sequence of characters.
- Index starts at 0.
String Operations

```
0 1 2 3 4 5 6 7 8 9 10 11
aStr:   h e l l o    w o r l d    !
```

aStr.length() is 12

Useful String Methods

- length() – Number of characters
**String Operations**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>aStr</code>:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>h</td>
<td>e</td>
<td>l</td>
<td>l</td>
<td>o</td>
<td>w</td>
<td>o</td>
<td>r</td>
</tr>
</tbody>
</table>

`aStr.charAt(4)` is 'o'

**Useful String Methods**

- `length()` – Number of characters
- `charAt(index)` – Character at given index
**String Operations**

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>aStr:</td>
<td>h</td>
<td>e</td>
<td>l</td>
<td>l</td>
<td>o</td>
<td></td>
<td>w</td>
<td>o</td>
<td>r</td>
<td>l</td>
<td>d</td>
<td>!</td>
</tr>
</tbody>
</table>
```

`aStr.substring(1,7)` is "ello w"

**Useful String Methods**

- `length()` – Number of characters
- `charAt(index)` – Character at given index
- `substring(startIndex,endIndex)` – Sub-string from `startIndex` to `endIndex - 1`
**String Operations**

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
</table>
```

```
aStr: h e l l o w o r l d !
```

```
aStr.indexOf('l') is 2
```

**Useful String Methods**

- `length()` – Number of characters
- `charAt(index)` – Character at given index
- `substring(startIndex, endIndex)` – Sub-string from `startIndex` to `endIndex - 1`
- `indexOf(item, startIndex)` – Returns the first index of `item` from `startIndex`
String Operations

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>aStr:</td>
<td>h</td>
<td>e</td>
<td>l</td>
<td>l</td>
<td>o</td>
<td>w</td>
<td>o</td>
<td>r</td>
<td>l</td>
<td>d</td>
<td>!</td>
</tr>
</tbody>
</table>

aStr.indexOf('l', 5) is 9

Useful String Methods

- `length()` – Number of characters
- `charAt(index)` – Character at given index
- `substring(startIndex, endIndex)` – Sub-string from startIndex to endIndex - 1
- `indexOf(item, startIndex)` – Returns the first index of item from startIndex
**String Operations**

```
aStr:  h e l l o  w o r l d  !
```

\`aStr.indexOf(’?’)\` is -1

**Useful String Methods**

- `length()` – Number of characters
- `charAt(index)` – Character at given index
- `substring(startIndex,endIndex)` – Sub-string from `startIndex` to `endIndex - 1`
- `indexOf(item, startIndex)` – Returns the first index of `item` from `startIndex`
**String Operations**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>aStr:</td>
<td>h</td>
<td>e</td>
<td>l</td>
<td>l</td>
<td>o</td>
<td>w</td>
<td>o</td>
<td>r</td>
<td>l</td>
<td>d</td>
<td>!</td>
</tr>
</tbody>
</table>

aStr.indexOf("ll") is 2

**Useful String Methods**

- `length()` – Number of characters
- `charAt(index)` – Character at given index
- `substring(startIndex, endIndex)` – Sub-string from `startIndex` to `endIndex - 1`
- `indexOf(item, startIndex)` – Returns the first index of `item` from `startIndex`
**String Operations**

```
0 1 2 3 4 5 6 7 8 9 10 11
```  
```
aStr: hello world!
```  
```
aStr.lastIndexOf('l') is 9
```  
*Useful String Methods*

- `length()` – Number of characters
- `charAt(index)` – Character at given index
- `substring(startIndex, endIndex)` – Sub-string from `startIndex` to `endIndex - 1`
- `indexOf(item, startIndex)` – Returns the first index of `item` from `startIndex`
- `lastIndexOf(item, startIndex)` – Reverse of `indexOf`
Strings and Scanner


Hello World!

```
hello world!
```

Tokenizing

delimiter – Sequence of characters (1 or more) that separate the regions of interest.

token – Regions of interest.

whitespace – space, tab (\t), newline (\n), carriage return (\r), form feed (\f).
Strings and Scanner

Strings and Scanner


Delimiter: Any amount of whitespace

Tokenizing

delimiter – Sequence of characters (1 or more) that separate the regions of interest.

token – Regions of interest.

whitespace – space, tab (\t), newline (\n), carriage return (\r), form feed (\f).
Strings and Scanner


```
sc

hello world!
```

Delimiter: Any amount of whitespace

Some more Scanner Instance Methods

Scanner sc = new Scanner(...);

- `sc.nextLine()` – Read until end of line.
- `sc.next()` – Default delimiter: Any amount of whitespace.
- `sc.useDelimiter(delim)` – Sets the delimiter to delim string.
- `sc.reset()` – Reset the delimiter to the default.
Strings and Scanner


```
sc.nextLine()
```

Delimiter: Any amount of whitespace

Some more Scanner Instance Methods

Scanner sc = new Scanner(...);

- `sc.nextLine()` – Read until end of line.
- `sc.next()` – Default delimiter: Any amount of whitespace.
- `sc.useDelimiter(delim)` – Sets the delimiter to delim string.
- `sc.reset()` – Reset the delimiter to the default.
Strings and Scanner


```java
sc
```

```plaintext
hello world!
```

```java
sc.nextLine()
sc.next()
```

Delimiter: Any amount of whitespace

### Some more Scanner Instance Methods

Scanner `sc = new Scanner(...);`
- `sc.nextLine()` – Read until end of line.
- `sc.next()` – Default delimiter: Any amount of whitespace.
- `sc.useDelimiter(delim)` – Sets the delimiter to delim string.
- `sc.reset()` – Reset the delimiter to the default.
Strings and Scanner


```
hello world!
```

Delimiter: Any amount of whitespace

Some more Scanner Instance Methods

Scanner `sc` = new Scanner(...);

- `sc.nextLine()` – Read until end of line.
- `sc.next()` – Default delimiter: Any amount of whitespace.
- `sc.useDelimiter(delim)` – Sets the delimiter to `delim` string.
- `sc.reset()` – Reset the delimiter to the default.
Strings and Scanner


Delimiter: Any amount of whitespace

Some more Scanner Instance Methods

Scanner sc = new Scanner(...);

  ● sc.nextLine() – Read until end of line.
  ● sc.next() – Default delimiter: Any amount of whitespace.
  ● sc.useDelimiter(delim) – Sets the delimiter to delim string.
  ● sc.reset() – Reset the delimiter to the default.
**Strings and Scanner**


```java
Scanner sc = new Scanner(...);
- sc.nextLine() – Read until end of line.
- sc.next() – Default delimiter: Any amount of whitespace.
- sc.useDelimiter(delim) – Sets the delimiter to delim string.
- sc.reset() – Reset the delimiter to the default.
```

Delimiter: A single "l" (TopHat Q4: How many tokens?)

```
hello world!
```

Strings and Scanner


Delimiter: A single "l"

Some more Scanner Instance Methods

Scanner sc = new Scanner(...);

- sc.nextLine() – Read until end of line.
- sc.next() – Default delimiter: Any amount of whitespace.
- sc.useDelimiter(delim) – Sets the delimiter to delim string.
- sc.reset() – Reset the delimiter to the default.
Instance vs Static Method

Instance Method

A method (non-static) that requires an instance of the class.

\texttt{instanceName.methodName(...)}

Ex.

\begin{verbatim}
String s = "Hello World!"
s = s.substring(0,3);
\end{verbatim}
Instance vs Static Method

Instance Method

A method (non-static) that requires an instance of the class.

\[ \text{instanceName} \text{. methodName}(\ldots) \]

Ex.

\begin{align*}
\text{String } s &= \text{"Hello World!"} \\
\text{s} &= \text{s.substring(0,3)};
\end{align*}

Static Method

A (static) method that does not require an instance of the class to run.

\[ \text{ClassName} \text{. methodName}(\ldots) \]

Ex.

\[ \text{Math.pow}(2,5); \]
Back to References
**Immutable Objects**

String and the primitive wrappers are *immutable*.

**Definition**

An immutable object is an object that cannot be modified after its creation.

Diagram:

Stack

Heap
**Immutable Objects**

String and the primitive wrappers are *immutable*.

**Definition**

An immutable object is an object that cannot be modified after its creation.

```
Integer i = 6;
```
**Immutable Objects**

String and the primitive wrappers are *immutable*.

**Definition**

An immutable object is an object that cannot be modified after its creation.

```
Integer i = 6;
Integer j = 6;
```
**Immutable Objects**

String and the primitive wrappers are *immutable*.

**Definition**

An immutable object is an object that cannot be modified after its creation.

```java
Integer i = 6;
Integer j = 6;
j = 5;
```
Immutable Objects

String and the primitive wrappers are immutable.

**Definition**

An immutable object is an object that cannot be modified after its creation.

```
Integer i = 6;
Integer j = 6;
j = 5;
Integer k = new Integer(5);
```
**Immutable Objects**

String and the primitive wrappers are *immutable*.

**Definition**

An immutable object is an object that cannot be modified after its creation.

```
String s = "foobar";
```
**Immutable Objects**

String and the primitive wrappers are *immutable*.

**Definition**

An immutable object is an object that cannot be modified after its creation.

```java
String s = "foobar";
s = s.substring(0, 3);
```
Random Numbers
TopHat Question 5

Pick the random binary string:

- 111111111111111111111111111111111111111111111
- 110011010110110101100101100101110101001010111
- 1100100100001111101010100010000101101000
- 1100100100001111101101010100010001000101101000
What is randomness?

Tour of Accounting

Over here we have our random number generator.

Nine nine nine nine nine nine

Are you sure that's random?

That's the problem with randomness: you can never be sure.

www.dilbert.com scottadams@aoi.com

©Asl, 2001 United Feature Syndicate, Inc.
What is randomness?

Anecdotally, humans are biased towards sequences without discernible patterns.
What is randomness?

Anecdotally, humans are biased towards sequences without discernible patterns.

Kolmogorov Complexity

How succinctly can the sequence be described?
What is randomness?

Anecdotally, humans are biased towards sequences without discernible patterns.

Kolmogorov Complexity

How succinctly can the sequence be described?

- 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 : 48 1’s (Low)
- 1100110101101101011001011001011110101001010111 : no clue (High)
- 11001001000011111101101010100010001000101101000 : \( \pi \) in binary to 46 floating-point digits (Low)
Sources of Randomness

Lack of Knowledge

We don't know the forces being applied.
Sources of Randomness

Lack of Knowledge
We don’t know the forces being applied.

Certified Random Bits
It is possible to get truly random bits via the quantum world:

Random numbers certified by Bell’s theorem
S. Pironio, A. Acín, S. Massar, A. Boyer de la Giroday, D. N. Matsukevich, P. Maunz, S. Olmschenk, D. Hayes, L. Luo, T. A. Manning & C. Monroe
Nature 464, 1021-1024 (15 April 2010)
Sources of Randomness in Java

Pseudorandom Number Generator

A pseudorandom number generator is able to produce a sequence of seemingly random numbers.
Sources of Randomness in Java

Pseudorandom Number Generator

A pseudorandom number generator is able to produce a sequence of seemingly random numbers.

- The first random value \( r_1 \) is calculated from a seed.
Sources of Randomness in Java

Pseudorandom Number Generator

A pseudorandom number generator is able to produce a sequence of seemingly random numbers.

- The first random value $r_1$ is calculated from a *seed*.
- Typically, the seed will be based on the current time.
Sources of Randomness in Java

Pseudorandom Number Generator

A pseudorandom number generator is able to produce a sequence of seemingly random numbers.

- The first random value $r_1$ is calculated from a seed.
- Typically, the seed will be based on the current time.
- All subsequent random numbers are calculated from the previous value.
  (I.e. $r_n = f(r_{n-1})$)
**Sources of Randomness in Java**

**Pseudorandom Number Generator**

A pseudorandom number generator is able to produce a sequence of seemingly random numbers.

- The first random value \( r_1 \) is calculated from a *seed*.
- Typically, the seed will be based on the current time.
- All subsequent random numbers are calculated from the previous value.
  (I.e. \( r_n = f(r_{n-1}) \))
- Pseudorandom because the seed fully determines the sequence of numbers.
**Random Class**

Standard Java class: `import java.util.Random;`

**Important Instance Methods**

**Random Class**

Standard Java class: `import java.util.Random;`

**Important Instance Methods**

- **Constructor:** `Random rand = new Random();` – "Random" seed.
- **Constructor:** `Random rand = new Random(seed);` – Fixed seed.
**Random Class**

Standard Java class: `import java.util.Random;`

**Important Instance Methods**

- Constructor: `Random rand = new Random(seed);` – Fixed seed.
- `setSeed(seed)` – Resets the instance to a fixed seed.
**Random Class**

Standard Java class: `import java.util.Random;

**Important Instance Methods**

- Constructor: `Random rand = new Random(seed);` – Fixed seed.
- `setSeed(seed)` – Resets the instance to a fixed seed.
- `nextInt(bound)` – Returns a pseudorandom number between 0 and `bound - 1`. 
Random rand = new Random();
int randInt = rand.nextInt(10) + 1;
**Using Random**

```java
Random rand = new Random();
int randInt = rand.nextInt(10) + 1;
```

Between 0 and 9
**Using Random**

```java
Random rand = new Random();
int randInt = rand.nextInt(10) + 1;
```

Between 0 and 9  

Shifts by 1
Complete the following code snippet so that `randInt` gets a pseudorandom number between 3 and 18 inclusive:

```java
Random rand = new Random();
int randInt = 20/21
```
Further Reading

COMP SCI 200: Programming I
zyBook code: WISCCOMPSCl200Fall2017

- Chapter 3. Using Objects
Image Sources I

http://dilbert.com/

http://www.clinovo.com/userfiles/clincapture/randomization.jpg

https://brand.wisc.edu/web/logos/

http://www.zybooks.com/