2 Language Basics – Variables

- Array is itself a reference type:

```java
int[] someArray = new int[3];
int[] anotherArray = new int[3];
someArray[2] = 7;
anotherArray[1] = 8;
```

- Array: "Indexed list of elements"
- Holds a fixed number of variables of a certain type (primitive or reference)
- Is itself a reference type (see next slide)

```java
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```
2 Language Basics – Variables

Arrays

- Array is itself a reference type:

```java
int[] someArray = new int[3];
int[] anotherArray = new int[3];
someArray[2] = 7;
anotherArray[1] = 8;
someArray = anotherArray;
boolean b = (someArray.length == 8);
// b == true

- Length property:

```java
int l = someArray.length;
// l == 3
```

2 Language Basics – Expressions, Statements, Blocks

Expressions

- Expression: Legal combination of constants, variables and operators
- Can be (and typically are) nested
- Expressions evaluate to a value of a certain type

Given:

```java
int a = 73;
boolean someArray[] = new boolean[5];
```

<table>
<thead>
<tr>
<th>Example</th>
<th>Evaluates to</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>int</td>
</tr>
<tr>
<td>2.0 / 3.0</td>
<td>0.6666666666</td>
<td>double</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>boolean</td>
</tr>
<tr>
<td>15 / 8</td>
<td>1</td>
<td>int</td>
</tr>
<tr>
<td>(17 + (3 * 9)) % 3</td>
<td>2</td>
<td>int</td>
</tr>
<tr>
<td>a + 1</td>
<td>74</td>
<td>int</td>
</tr>
<tr>
<td>a * 9.0 / someArray.length</td>
<td>131.4</td>
<td>double</td>
</tr>
</tbody>
</table>

2 Language Basics – Expressions, Statements, Blocks

Expressions

- Some expressions have so-called side-effects

Given:

```java
int a = 73; int b;
```

<table>
<thead>
<tr>
<th>Example</th>
<th>Value</th>
<th>Side-effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = 84</td>
<td>84</td>
<td>Assign 84 to a</td>
</tr>
<tr>
<td>b = (a = 48)</td>
<td>48</td>
<td>Assign 48 to both a and b</td>
</tr>
<tr>
<td>a++</td>
<td>48</td>
<td>Assign 49 to a (a++)</td>
</tr>
<tr>
<td>++a</td>
<td>50</td>
<td>Assign 50 to a (a++)</td>
</tr>
<tr>
<td>new Bicycle()</td>
<td>Reference to the new instance of Bicycle, e.g. &lt;1150&gt;</td>
<td>Create and initialize new instance of class Bicycle in memory</td>
</tr>
<tr>
<td>new double[10]</td>
<td>Reference to the new array of double</td>
<td>Create and initialize new array in memory</td>
</tr>
</tbody>
</table>

2 Language Basics – Expressions, Statements, Blocks

Statements

- Statement: Complete unit of execution (ends with `;`)
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```java
void applyBrakes()
{
    if (speed > 0) {
        speed = speed - 1;
    }
}

void applyBrakes()
{
    if (speed > 10) {
        speed = speed - 2; // break really hard
    } else if (speed > 0) {
        speed--; // soft brakes
    } else {
        System.err.println("The bicycle has already stopped!");
    }
}
```

- switch: Equivalent to sequence of chained if else statements

- while: do something as long as some condition (boolean expression) is true

```java
int count = 1;
while (count < 8) {
    System.out.print("#" + count + ",");
    count++;
}
```

- output will be: #1 #2 #3 #4 #5 #6 #7

- do while: similar to "while", but check condition at the end of execution of something instead of at the beginning

```java
int count = 1;
do {
    System.out.print("#" + count + ",");
    count++;
} while (count < 8);
```

- output will be: #1 #2 #3 #4 #5 #6 #7

2 Language Basics – Control Flow Statements

- for: usually means to do something for a fixed number of times:

```java
for (int i=0; i<7; i++) { // loop will be executed 7 times
    System.out.print("#" + i + ",");
}
```

- General form:

```java
for (initialization; termination; update) {
    statement
}
```

- initialization expression: Executed once at the beginning of first loop
- termination expression: if true then execute statement(s), else exit loop
- update expression: Executed after each iteration of the loop

2 Language Basics – Control Flow Statements

- for: usually means to do something for a fixed number of times:

```java
for (int i=0; i<7; i++) { // loop will be executed 7 times
    System.out.print("#" + i + ",");
}
```

- General form:

```java
for (initialization; termination; update) {
    statement
}
```

- initialization expression: Executed once at the beginning of first loop
- termination expression: if true then execute statement(s), else exit loop
- update expression: Executed after each iteration of the loop
3 Classes, Objects, Inheritance

Deepening readings:

http://java.sun.com/docs/books/tutorial/java/javaOO/classes.html
http://java.sun.com/docs/books/tutorial/java/javaOO/objects.html
http://java.sun.com/docs/books/tutorial/java/javaOO/more.html
http://java.sun.com/docs/books/tutorial/java/lang/subclasses.html

2 Language Basics – Control Flow Statements

- **for:** usually means to do something for a fixed number of times:

  ```java
  for (int i=0; i<7; i++) { // loop will be executed 7 times
    System.out.print("#" + i + ");
  }
  ``

  → output will be: #:0 #:1 #:2 #:3 #:4 #:5 #:6

- General form:

  ```java
  for (initialization; termination; update) {
    statement
  }
  ```

  - **initialization** expression: Executed once at the beginning of first loop
  - **termination** expression: If true then execute statement(s), else exit loop
  - **update** expression: Executed after each iteration of the loop

2 Language Basics – Expressions, Statements, Blocks

- Variables **declared inside** a block are only visible from within that block:

  ```java
  int a = 7, b = 6;
  if (a != b) { // begin block
    int c;
    c = a * b;
    System.out.println(c);
  } // end block
  System.out.println(c); // ERROR: c unavailable
  ```
3 Classes, Objects, Inheritance

Deepening readings:

http://java.sun.com/docs/books/tutorial/java/javaOO/classes.html
http://java.sun.com/docs/books/tutorial/java/javaOO/objects.html
http://java.sun.com/docs/books/tutorial/java/javaOO/more.html
http://java.sun.com/docs/books/tutorial/java/land/subclasses.html
3 Classes, Objects, Inheritance

- Field declaration (general form):
  \[ \text{modifier type name}; \]
- (Access) modifier (for fields):
certain combinations of \{public, protected, private, static, final\}
- type: Any primitive or reference type

```java
class Bicycle {
  public int cadence = 0;
  public int speed = 0;
  public int gear = 1;

  // Field declaration
  public int seatHeight;

  public Bicycle(int startCadence, int startSpeed, int startGear) {
    super(startCadence, startSpeed, startGear);
    seatHeight = startHeight;
  }
}
```

- Method declaration (general form):
  \[ \text{modifier returnType name (parameter*) throwsClause\{statement*\}} \]
- (Access) modifier (for methods):
certain combinations of \{public, protected, private, static, final, abstract\}
- returnType: Any primitive or reference type
- parameter*: (later)
- throwsClause*: (later)
- statement*: statement(s) to execute

```java
public void setHeight(int newValue) {
  seatHeight = newValue;
}
```

- Constructor declaration (general form):
  \[ \text{modifier MyClass (parameter*) throwsClause\{statement*\}} \]
- (Access) modifier:
certain combinations of \{public, protected, private\}
- parameter*: (later)
- throwsClause*: (later)
- statement*: statement(s) to execute

```java
public Bicycle(int c, int s, int g) {
  cadence = c;
  speed = s;
  gear = g;
}
```

- Why do we need constructors?
  - Ensure complete and consistent initialization after object creation
  - Access (non-default) superclass constructors:
    Construct object according to definition of superclass, then add specifics
  - Provide several constructors for varying use-cases

```java
class Tandem extends Bicycle {
  public int numberOfDrivers;

  public Tandem(int c, int s, int g, int n) {
    super(c, s, g);
    numberOfDrivers = n;
  }
}
```
3 Classes, Objects, Inheritance

Why do we need constructors?

- Ensure complete and consistent initialization after object creation
- Access (non-default) superclass constructors: Construct object according to definition of superclass, then add specifics
- Provide several constructors for varying use-cases

```java
class Bicycle {
    public int cadence;
    public int speed;
    public int gears;
    public Bicycle(int c, int s, int g) {
        cadence = c;
        speed = s;
        gear = g;
    }
    public Bicycle(int g) {
        cadence = 0;
        speed = 0;
        gear = g;
    }
}
class Tandem extends Bicycle {
    public int numberOfDrivers;
    public Tandem(int c, int s, int g, int n) {
        super(c, s, g);
        numberOfDrivers = n;
    }
}
```

Example:

```java
class Person {
    String firstName;
    String lastName;
    long taxId;
    Person(String fName, String lName) {
        firstName = fName;
        lastName = lName;
        taxId = createUniqueTaxIdentifier(); // side-effect!
    }
    class Student extends Person {
        long matricNo;
        Student(String fName, String lName, long matric) {
            super(fName, lName);
            matricNo = matric;
        }
    }
    // Manual initialization, easy to make a mistake (e.g. what about 'taxId'?)
    Person student1 = new Person("Max", "Mustermann");
    Person student2 = new Person("Max", "Mustermann");
    Person student3 = new Person("Max", "Mustermann");
    // Complete, consistent, convenient
    Student student2 = new Student("Max", "Mustermann", 1234567890);
    Student student2 = new Student("Max", "Mustermann", 1234567890);
}```