

# Java Objects and Classes – the basics

---

# Java Objects and Classes – the basics

---

- Class: collection of data declarations and methods.
  - often a class models the attributes and actions associated with a *type* of real world *thing*.
- In addition to data and methods, a Class has
  - name
  - protections ("public" vs. "private", etc).
  - relationships to other classes (and a few other things).

# A Class definition defines only a type!

---

- A class does not *contain* any data, only definitions of data.
- An Object is an *instance* of a Class.
  - each object contains instance variables that correspond to the data declarations of the Class.
- Example:
  - Class `Person` declared to have name and age.
  - Object `Joe` (of type "`Person`") has name `joe` and age `22`.

# Java vs. C++

---


- When you define a Class in java, you include all the code for the methods in the file.
  - There are no include files in java.
- For C++ programmers, everything else will seem very familiar.

# A Class Example

---

```
class Person {  
    String name;  
    int age;  
  
    Person(String n, int a) {  
        name = n;  
        age = a;  
    }  
}
```

*Constructor*



# Creating an object

---

- Declaring a variable does not create an object!
- You must use the new operator to create an object (or call a method that creates an object – more on this later...).
- Example:

```
Person p = new Person("Fred", 22);
```

# Reference Variables

---

```
Person p;
```

- declares a variable of type "Person".
- Initially `p` does not refer to anything...
  - you have not created a Person object!

```
p = new Person("fred", 22);
```

- now `p` refers to something.
- You can (try to) print it:

```
System.out.println(p);
```

# A Test *program*

---

```
class TestPerson {  
    public static void main(String[] args) {  
        Person p = new Person("joe",22);  
        System.out.println(p);  
    }  
}
```

```
> java TestPerson  
Person@82ba41
```

# Another Test *program*

---

```
class TestPerson {  
    public static void main(String[] args) {  
        Person p;  
        System.out.println(p);  
    }  
}
```

```
> javac TestPerson.java  
TestPerson.java:4: variable p might not have  
    been initialized  
        System.out.println(p);  
                        ^  
1 error
```

# Add a method to Person class

---

```
class Person {
    String name;
    int age;

    Person(String n, int a) {
        name = n;
        age = a;
    }

    public String toString() {
        return name + " (" + age + ")";
    }
}
```

# New test code

---

```
class TestPerson {  
    public static void main(String[] args) {  
        Person p = new Person("joe",22);;  
        System.out.println(p.toString());  
    }  
}
```

```
> java TestPerson  
joe (22)
```

```
public String toString()
```

---

- This is called (automatically) whenever Java needs to turn an object into a String.

```
System.out.println(p.toString());
```

```
System.out.println(p);
```

*these generate  
the same output!*



- You could also do this:

```
System.out.println("Person is" + p);
```

# default toString ( )

---

- By default, a class has a `toString` method that generates the class name followed by "@" followed by the object's address:

`Person@82ba41`

# Quizarooni: What will this do?

---

```
class Quizarooni {
    String something;

    Quizarooni (String s) {
        something = s;
    }

    public int toString() {
        return (something.length());
    }

    public static void main (String args[]) {
        Quizarooni q = new Quizarooni (args[0]);
        System.out.println (q);
    }
}
```

*Notice the built-in test code!*

# Answerooni: nothing, it won't compile

---


```
class Quizarooni {
    String something;

    Quizarooni (String s) {
        something = s;
    }

    public int toString() {
        return (something.length());
    }

    public static void main (String args[]) {
        Quizarooni q = new Quizarooni (args[0]);
        System.out.println (q);
    }
}
```

*Needs to return String!*



# Why does it need to return String?

---

- Inheritance!
  - When a class is a *special version* of another class called the *base class*.
  - The class *inherits* methods and fields from the base class.
  - If the base class has a method named `foo()` that returns a `String`, we can't replace this with a method that returns something other than a `String`.

# What is the base class in Quizarooni?

---

- All java classes are special versions of the class **Object**
  - **Object** is the base class of all classes
    - might actually be the base class of the base class of a class...
  - class **Object** has a method with the *prototype*:  
`String toString()`

# Implications

---

- Every class has a **toString()** method.
  - could be supplied by the Object class, or could be supplied by the class itself.
- What else does Object have?
  - Methods include:
    - `clone()`
    - `getClass()`
    - `equals(Object o)`
  - look up `java.lang.Object` in the API docs!

# Subtle Issue

---

- Is this legal?

```
public int toString(int x) {  
    return (something.length() + x);  
}
```

- Yes. The argument list to the function is different.
- BUT – this is not the method that will be called automatically by `System.out.println()`.

# OK, perhaps not so subtle.

---

- what's the difference between:
  - different return types
  - different argument lists.
- If *only* the return type is different, how does the compiler know which version you are calling?
  - It can't know. So – it's not allowed (this is also the case in C++).
- Two methods of the same class must have different names, or the same name but different arguments.

# Protection

---

- You can restrict access to classes, methods and/or data members.
  - This is a software engineering issue.
- We will explore in more detail later...
- Levels of protection (access modifiers):
  - **public** (any other class can access)
  - **private** (only this class)
  - **protected** (only classes in the same *package* and any *derived* classes)

# default protection

---

- The default is "any class in the same *package* can access it (*it* could be a class, method or field).
- Ok, but what the heck is a *package*?
  - roughly: a collection of classes that have been declared to be part of the same package...
  - For now – all Classes in the same directory/folder are in the same package unless we do something to prevent this.

# Getting started

---

- create a bunch of classes in the same directory/folder.
- don't include any access modifiers
- All methods will be able to access all methods and fields of other classes.

# Example: Person and TestPerson

---

```
class Person {
    String name;
    int age;

    Person(String n, int a, String) {
        name = n;
        age = a;
    }

    public String toString() {
        return name + " (" + age + ")";
    }
}
```

# Example: Person and TestPerson

---

```
class TestPerson {  
  
    public static void main(String[] args) {  
        Person p = new Person("joe", 22);  
        System.out.println("Person is " + p);  
    }  
  
}
```

# Quiz-ewski

---

```
class TestPerson {
    public static void main(String[] args) {
        for (int i=0;i<args.length;i++) {
            TryPerson(args[i]);
        }
    }
    void TryPerson(String name) {
        Person p = new Person(name,19);
        System.out.println(p);
    }
}
```

# Won't Compile-ewski

---

```
class TestPerson {  
    public static void main(String[] args) {  
        for (int i=0;i<args.length;i++) {  
            TryPerson(args[i]);  
        }  
    }  
  
    void TryPerson(String name) {  
        Person p = new Person(name,19);  
    }  
}
```

```
TestPerson2.java:5: non-static method  
TryPerson(java.lang.String)  
    cannot be referenced from a static context  
        TryPerson(args[i]);  
        ^
```

```
1 error
```

# Static Methods

---

- A static method is basically like a function
  - you don't need an object to call the method.
  - there is no object when the method is called!
- Static methods can't:
  - access any fields
  - call any non-static methods
- `main` is static
  - the egg comes before the chicken.

# Revised TestPerson

---

```
class TestPerson2 {
    public static void main(String[] args) {
        TestPerson2 x = new TestPerson2();
        for (int i=0;i<args.length;i++) {
            x.TryPerson(args[i]);
        }
    }

    void TryPerson(String name) {
        Person p = new Person(name,19);
        System.out.println(p);
    }
}
```

# Some important Classes

---

- For each primitive type there is a corresponding Class.
  - *wraps* a primitive in an object.
  - The class name starts with a capital letter.

**Integer**

**Short**

**Byte**

**Long**

**Char**

**Double**

**Float**

**Boolean**

*these are in java.lang*

# What can a *wrapper* do for me?

---

- Parse a string:
  - `static int Integer.parseInt(String) ;`
  - `static boolean Boolean.parseBoolean(String) ;`
  - ...
- Some type conversions:
  - `double Integer.doubleValue() ;`
  - `long Integer.longValue() ;`
  - ...

# Wrapper Constructors

---

```
Integer(int i)    Integer(String s)
Double(double d) Double(String s)
...

```

- Example use:

```
Integer foo = new Integer(12);
```

```
Double pancake = new Double("3.14159");
```

```
Boolean booley = new Boolean(true);
```

# AutoBoxing

---

- This is new! (needs Java 1.5).
- Autoboxing: you can assign a primitive type value to wrapper object directly.

```
Integer x = 17;
```

- You used to have to do this:

```
Integer x = new Integer(17);
```

- I guess someone on the Java Language Committee has arthritis...

# Unboxing

---

- You can also assign a wrapper object as the value of a primitive type variable
  - or use a wrapper object anyplace you would normally use a primitive type.

```
Integer foo = new Integer(27) ;
```

```
int i = foo + 7 ;
```

- You used to do this:

```
int i = foo.intValue() + 7 ;
```

# Autoboxing/unboxing

---

- There is no performance improvement, this is completely a compiler issue.
  - The compiler generates the extra code for you, so it still takes just as long at runtime.
- If you are used to doing things *the old way*, things will still work fine.
  - and I have an 8 track tape player I'd like to sell you. I'll throw in a couple of Disco tapes for free.