Dynamic Types

…Assignment to a subtype

- If public Duck extends Bird { …
- Then, you may code:

```
    ….
    Bird bref;
    Duck quack = new Duck();
    bref = quack;
}
```

A subtype may be assigned where the supertype is expected

…Assignment to a subtype

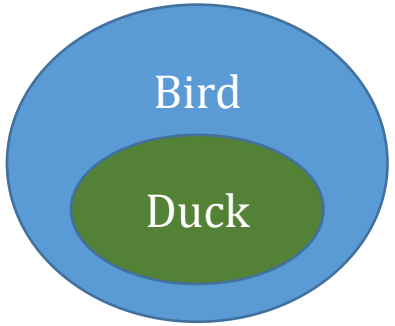
- If public Duck extends Bird { …
- Then, you may code:

```
    ….
    Bird bref;
    Duck quack = new Duck();
    bref = quack;
}
```

Static Type: bref is declared as a reference to a Bird object

Dynamic Type: bref is a reference to a Duck object!

Implicit Up-cast



- Explicit Cast : Treat a Duck object as if it were a Bird object

```
Bird bref = (Bird) quack;
```

- Up-cast : Cast of a sub-type to a super-type ("widening")
  - Always legal because sub-type (Duck) "contains" super-type (Bird)
  - "Throws away" extra sub-type (Duck) fields and methods

- Implicit Up-cast : Compiler performs the widening up-cast without direction from the programmer

```
Bird bref = quack;
```

No compiler messages!

Static Type

- Evaluated at compile time
- Specified by the programmer
- Used by the compiler to issue warning and error messages
  - Compiler only *allows* invocation of methods based on static type

```
Bird bref;
Duck quack = new Duck();
bref = quack;
bref.swim();
```

Birds do not have a swim method

Ducks can swim

Implicit up-cast to Bird

Compiler error: Birds can't swim!

Dynamic Type

- Evaluated at run-time
- Java keeps track of the ACTUAL type of a reference, even when the reference has been implicitly up-cast to a super-type.
- The type specified after "new"

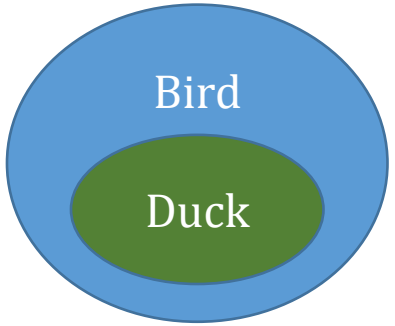
```
Duck quack = new Duck();
Bird bref = quack;
```

Static type: Duck
Dynamic type: Duck

Static type: Bird
Dynamic type: Duck

- Dynamic type is used to select the method!
  - Enables polymorphism!
- Dynamic type checked at run time when explicit down-cast occurs

Static Down-cast



- Explicit Cast : Treat a Bird object as if it were a Duck object

```
Duck dref = (Duck) bref;
```

- Down-cast : Cast of a super-type to a sub-type ("narrowing")
  - Not always legal because super-type (Bird) may not be a sub-type (Duck)
  - Only legal if the dynamic type of bref is Duck
    (Or dynamic type of bref is a sub-type of Duck)
  - "Restores" extra sub-type (Duck) fields and methods to dref

- No Implicit Down-cast : Compiler will not perform the narrowing down-cast without direction from the programmer

```
Duck dref = bref;
```

Compiler error: bref is not a Duck

Static Down-casting

- Requires sophisticated programmer to get it right
- Programmer tells compiler: "Even though you think this is the super-type, I know that it's really the sub-type."
- If the programmer is wrong, run-time errors occur!
- Use instanceof to ensure you've got it right

```
if (bref instanceof Duck) {
        Duck dref = (Duck) bref;
```

Casting and Operator Precedence

- Cast occurs before normal operators

```
(double) sum / array.length;
```

- But "." occurs before cast

```
(double) array.length
```

- Extra parenthesis required to override "."

```
(Alpha)var1.mk();   // cast result of var1.mk() to Alpha
((Alpha)var1).mk(); // cast var1 to Alpha
// Invokes Alpha class mk method
```

Static AND Dynamic Types

- You must pay attention to both static and dynamic types
- Static types are used by the compiler to determine whether instructions are legal
- Dynamic types are used for method dispatch