Higher-order programming

O3/J9

- Higher-order programming
- Interfaces
- Java lambda expressions

Higher-order programming

- Write functions (or classes) parameterised by functions.
  - Write functions that are more general, to reduce code repetition.
  - Callback methods (event-driven programs).
- In Java, higher-order functions are implemented with interfaces and anonymous classes or lambda expressions.

Reminder: Interfaces in Java

- Like a class, a Java `interface` defines a set of method signatures (return type, name and parameter types), but provides no implementation of the methods and has no fields.
- A class can implement one or more interfaces, by providing the required methods.
- An `interface` can be used as a type: a variable (parameter) of an interface type can hold a reference to an object of any class that implements the interface.

Functional interfaces

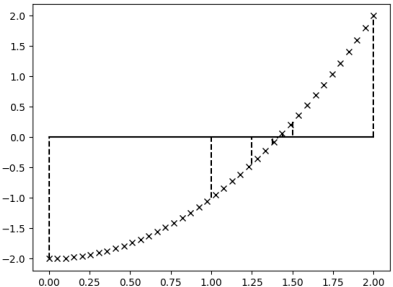
- An interface that defines a single method can be declared *functional*, using an annotation:

```
@FunctionalInterface
public interface RealValuedFunction {
  public double apply(double x);
}
```

- A functional interface can be implmented with a Java lambda expression.
- Common (generic) functional interfaces defined in the `java.util.function` package.

Example: Solving an equation

- Solve $g(x) = y$.
- Implement the interval-halving algorithm parameterised with g and y.



Reminder: lambda expressions in Java

- An anonymous implementation of a functional interface can be created with a lambda expression:

```
(double x) -> { return Math.pow(2, x); }
```

  - parameter list, in parentheses;
  - keyword “->”;
  - method body.
- can omit parameter types if the compiler can guess them from context (for example, from parameter type of the method that the lambda expression is passed to).